NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

CODE OPTIMIZATION OF SPECKLE REDUCTION ALGORITHMS FOR IMAGE PROCESSING OF ROCKET MOTOR HOLOGRAMS

by

Dana S. Kaeser

December 1988

Thesis Advisor: John P. Powers

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 62	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
CODE OPTIMIZATION OF SPECKLE REDUCTION ALGORITHMS FOR IMAGE PROCESSING OF ROCKET MOTOR HOLOGRAMS

12 PERSONAL AUTHOR(S)
KAESER, Dana S.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1988 December	88

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Virtual disk-based array; Speckle reduction algorithms; Code optimization; Holographic image processing

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis supplements and updates previous research completed in the digital analysis of rocket motor combustion chamber holographic images. In particular this thesis deals with the software code optimization of existing automatic data retrieval algorithms that are used to extract useful particle information from the holograms using a microcomputer-based imaging system. Two forms of optimization were accomplished, the application of an optimizing FORTRAN compiler to the existing FORTRAN programs and the complete rewrite of the programs in the C language using an optimizing compiler. The overall results achieved were a reduction in executable program size and a significant decrease in program execution speed.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL POWERS, John P.	22b TELEPHONE (Include Area Code) (408) 646-2081 22c OFFICE SYMBOL 62Po

DD Form 1473, JUN 86 *Previous editions are obsolete*

Approved for public release; distribution is unlimited.

CODE OPTIMIZATION OF SPECKLE REDUCTION ALGORITHMS FOR IMAGE PROCESSING OF ROCKET MOTOR HOLOGRAMS

by

Dana S. Kaeser
Lieutenant Commander, United States Navy
B.S., University of Mississippi, 1974

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

December 1988

Author: Dana S. Kaeser

Approved by: John P. Powers, Thesis Advisor

David W. Netzer, Second Reader

John P. Powers, Chairman,
Department of Electrical
and Computer Engineering

Gordon E. Schacher,
Dean of Science and Engineering

ABSTRACT

This thesis supplements and updates previous research completed in the digital analysis of rocket motor combustion chamber holographic images. In particular this thesis deals with the software code optimization of existing automatic data retrieval algorithms that are used to extract useful particle information from the holograms using a microcomputer-based imaging system. Two forms of optimization were accomplished, the application of an optimizing FORTRAN compiler to the existing FORTRAN programs and the complete rewrite of the programs in the C language using an optimizing compiler. The overall results achieved were a reduction in executable program size and a significant decrease in program execution speed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

VCR	Video Cassette Recorder
AOI	Area of Interest
VMB	Video Memory Board
AFRT	Air Force Resolution Target
SI	Speckle Index
IBM	International Business Machines
DOS	Disk Operating System
DIR	Directory
TSR	Temporary Stay Resident
MSFORT	Microsoft FORTRAN Ver. 4.1
MSC	Mirosoft C Compiler Ver. 5.0
STATGRAPHICS	Statistical Graphics Systems

I. INTRODUCTION

This thesis supplements and updates ongoing research at the Naval Postgraduate School. It documents and describes work completed to improve existing software algorithms used to process holographic images of rocket motor combustion products.

A. BACKGROUND

Holographic techniques are used to produce an image of the rocket motor combustion products so that a statistical analysis of the particles in the gases may be performed. The particles are from additives to the solid propellant. They are added for the purpose of improving certain performance characteristics of the rocket motor. The particle size distribution has a major effect on the performance of the motor, and it is this information that is sought in the research. A complete description of the hologram recording, reconstruction, and imaging process can be found in References 1, 2, and 3.

The main objective of this thesis was to improve the existing microcomputer software programs used to perform automatic data retrieval from the holograms. The area of research was directed mainly at reducing the executable code size and improving the time of execution of the programs. Two phases of improvement were attempted. The first was the application of optimizing compiler techniques to the existing FORTRAN programs. The second was a subsequent complete rewrite of the programs using the C programming language and an optimizing compiler.

B. HARDWARE DESCRIPTION

The hardware system consists of a standard IBM PC/AT computer with 40MB hard drive configured with an INTEL INBOARD 386/AT microprocessor with a 80387 math coprocessor

module, an AST ADVANTAGE memory expansion board with 1.5 megabytes of memory, and an Imaging Technology Inc. PCVISION Frame Grabber image processing board installed in an eight bit data bus slot of the IBM PC/AT. A detailed description of the PCVISION Frame Grabber hardware can be found in Reference 4. The external support items include a video monitor and VCR recorder for input, viewing, and recording of external images; an external 5.25 (20Mb) cartridge Bernouli disk drive used to store image files; and a Tektronix thermal copier used to get hardcopy of the displayed video monitor screen.

C. SYSTEM SOFTWARE

The software associated with the system includes the following:

- Imaging Technology's IMAGEACTION software package, a menu driven software package that provides numerous routines for real time processing of images [Ref. 5].
- Imaging Technology's ITEX/PC software library which contains the routines used in both the FORTRAN and C locally produced programs [Ref. 6].
- IBM Disk Operating System Version 3.3 (DOS).
- Microsoft FORTRAN Compiler Version 3.3.
- Microsoft FORTRAN Optimizing Compiler Version 4.1 (MSFORT) [Refs. 7, 8, and 9].
- Microsoft C Optimizing Compiler Version 5.0 (MSC) [Refs. 10, 11, and 12].
- Locally produced automatic data retrieval and analysis programs used to support the research [Refs. 1, 2, and 3].

D. THESIS ORGANIZATION

This thesis consists of four chapters. Chapter Two discusses the improvements made in the software systems and also gives a detailed discussion of the new programs developed using the C language. Chapter Three discusses the software and hardware performance analysis and shows the results of improvements made. A summary and concluding remarks are presented in Chapter Four. Appendices A through J contain the source code for the C

Language programs developed. These programs are heavily documented to explain their operation. Appendix K contains a description and files used to simplify and automate FORTRAN and C compiler operations.

II. SOFTWARE IMPROVEMENTS

Two phases of code optimization were conducted. The FORTRAN optimization was achieved using the MSFORT compiler version 4.1 [Ref. 7] and the C programming optimization was accomplished using MSC compiler version 5.0 [Ref. 10].

A. FORTRAN OPTIMIZATION

Prior to this thesis, the Microsoft FORTRAN Compiler Version 3.3 was used exclusively to compile locally produced programs. Previous programming done by Redman [Ref. 1], Edwards [Ref. 2], and Orguc [Ref. 3] used this compiler to compile all FORTRAN programs. One of the results of this thesis was the elimination of the dependence on this non-optimizing compiler.

The MSFORT optimizing compiler was available for use during the work done by Orguc but could not be used due to incompatibilities with the compiler and the ITEX/PC software library. The incompatibility was determined to be due to Microsoft's updating several aspects of the compiler to conform with the FORTRAN 77 ANSI standard. One of the improvements to the new compiler involved stringent enforcement of type casting between different modules in a program (i.e., main and subroutine modules). This type checking requirement resulted in several problems in the original programs due to type casts of variable declarations (i.e., CHARACTER *21) that were not consistent across main program and subroutine modules (i.e., main program declaration of CHARACTER *21 and the subroutine declaration of CHARACTER *22). This resulted in errors when compiling the programs but was not very evident when cross referencing the error code produced by the compiler. These type casts did not produce compile errors when compiling with the old compiler version.

The second problem dealt with errors generated within the ITEX/PC library FORTRAN include file *itexpc.inc* [Ref. 6] which is required to interface the frame grabber routines. The compiler

error code statement generated was *F2115: syntax error.* This error was attached to all array type declarations within the file. The error was traced to the format of the array declarations within the include file and was corrected by reversing the items within the declaration. A sample declaration before correction is

```
integer*2 jarray(9) [REFERENCE,NEAR],
```

with the correction being

```
integer*2 jarray[REFERENCE,NEAR] (9).
```

Once all of the syntax errors due to the above errors were corrected, the MSFORT compiler was usable with the ITEX/PC Software package.

B. C PROGRAMMING OPTIMIZATION

The C language optimization consisted of a complete rewrite of the locally developed FORTRAN routines using the MSC compiler. Several major achievements were realized using the C language. One significant improvement was the elimination of the need to define large data arrays for image pixel value storage. These data arrays used all the available memory assets (640 Kb DOS Limit) of the computer and resulted in the programs having to be written to use several loops or repeats of code to accomplish analysis of an entire image (512x480 pixels). In some cases the analysis of the entire image was not completed due to this restriction. The overall effects were mainly large-size executable programs that ran slower due to the extra overhead to compensate for these effects.

The C language allowed for the use of dynamic memory allocation during runtime and repeated calls to the functions of the ITEX/PC C library without a significant degradation in program speed or size. The use of a disk-based virtual array program also allowed for the easy use and access of large data arrays, when required, with minimum effect on the stringent memory restrictions. As a result the programs now support analysis of the entire usable image (512x480 pixels) with no special programming efforts to accomplish the tasks desired.

The following paragraphs describe the C programs generated for this thesis research. The performance and statistical analysis of these programs are presented in Chapter 3.

1. Program File *thesis.h*

The file *thesis.h* (Appendix A) is the header file for use with all the programs. It contains the required library include files necessary for successful compilation of the programs. It also serves to define the C manifest constants that are used throughout the programs. The main purpose for this header is to maintain a sense of portability among the constants and variable definitions used within the programs. It makes for easy redefinition of the constants if necessary without having to change these values wherever they appear within the code.

2. Program File *genfunc.c*

The general support functions program file *genfunc.c* (Appendix B) documents and lists a collection of functions used throughout the program files to perform routine evolutions. Each function title and comment within the appendix defines its purpose. Also included with this file is an array definition named *p[x][y]*. This array was used to test the programs offline before they were used on the image processing system. Use of the array is easy to implement and is explained in the comment statements located within the array definition.

3. Program File *threshit.c*

The Image threshold function program *threshit.c* (Appendix C) is the C language version of the threshold subroutine program presented by Edwards [Ref. 2]. This program takes an operator input value that is used as a threshold limit for the image and sets all pixel values below this value to BLACK (0) and those above it to WHITE (255). This effectively changes all pixel values representing image features to one value, thus producing a binary image of the original. This routine is used primarily to produce an image suitable for input to the feature identification program described later.

4. Program File *speckle.c*

Speckle noise index function, *speckle.c* (Appendix D), is a routine that uses the algorithm presented in Reference 13 and used by Edwards [Ref. 2] to calculate the value of speckle index. The value is a measure of the speckle noise present in an image. It is used to evaluate the effectiveness of a filter operation that has been performed to reduce the amount of speckle noise present in the image. This routine is used by all the filter routines to serve as a measure of their effectiveness.

5. Program File *vir_arry.c*

The virtual array support functions, file *vir_arry.c* (Appendix E), are a collection of the functions necessary to setup and access the virtual disk-based array scheme used in the filter programs discussed later. These functions were originally presented in Reference 15 and required only minor modification to support this research effort.

The access macro definitions and C type casts necessary to use the virtual array routines are defined in the program header file *thesis.h* (Appendix A). It is these definitions and macros that make the routines easy to use. When properly set up, the virtual disk array can be used as if it were the same as any C language array declared internal to a program file. All disk evolutions and file handling are done in the background without operator intervention. For a detailed analysis, or before modification of any of the functions, it is recommended that Reference 15 be consulted.

a. Function *init_v_array (filename, record_size, filchar)*

This function creates a virtual array file named *filename* on the default disk drive. The size of each element will be set to the value of *record_size*. The variable *filchar* is the character that is set into initial array values before formal assignment of the value is made. The number of elements in the array is set initially to zero and the file closed. This routine

provides the necessary file and initial settings to be used by the other functions called to use the virtual array. For example, a statement such as

```
init_v_array("pix.var",sizeof(int),NULL);
```

would create a file named *pix.var* that would have *integer* elements and use the defined *NULL* value as the fill character.

b. Function *open_v_array(filename,buffer_size)*

This function prepares an existing virtual array for use by opening the existing file of *filename*. It dynamically allocates memory space for a buffer to hold the elements of the array for immediate use. The size of this buffer is the value *buffer_size*. A statement such as

```
array = open_v_array("pix.var",100);
```

would open the file *pix.var* for use and create a buffer to hold 100 elements. The variable *array* must have been defined as a pointer variable.

c. Function *close_v_array(array)*

The function *close_v_array* writes any remaining elements from the buffer to disk, closes the virtual array file, and releases the allocated memory for the buffer. This routine should be the last one called in the program before exit.

d. Function *access_v_rec(array,index)*

This routine performs the low-level access to the file and makes sure the array element referenced by *index* is available in the memory buffer. If the element is not in the buffer, it reads the element from the disk. It also will extend the size of the virtual array automatically if the indexed value is not in the array. This allows for the elements of the array to be assigned only when needed and the array file only to grow as large as necessary to hold declared array values.

6. Program File *feat_id.c*

Program *feat_id.c* (Appendix F) is a complete stand-alone program that is used to scan the image seeking connected features and to assign a unique identification number to each

identified feature within an image. The maximum number of features that can be counted is limited only by the maximum value that the integer variable can assume (32,767). This number is then used to give a count of the number of features present in the image. The program must be supplied an image that has been thresholded since all decisions within the program are based on the value of the pixels being either WHITE or BLACK.

This program is a complete rewrite of the FORTRAN program presented by Orguc [Ref. 3]. It uses essentially the same program sequencing of events and ideas as the original, but the use of the GO TO statement that was used numerous times in the FORTRAN version to control program flow has been entirely eliminated from this version resulting in smoother program flow.

7. Program File *sizeit.c*

Image feature sizing is accomplished using program *sizeit.c* (Appendix G). This routine is a rewrite of the program presented by Orguc [Ref. 3]. It is a stand-alone program that takes the feature labeled output from the identification program (*feat_id.c*) and produces an output file listing each feature's area, x-chord, and y-chord. The *size.dat* output file is in a form suitable for input into a statistical analysis program such as STATGRAPHICS [Ref. 14]. This file must be saved under another filename or processed into STATGRAPHICS before the sizing program is executed again because any existing file is overwritten during program execution. The main new feature of the C language version over the FORTRAN is that dynamic memory allocation was used to allocate data storage for each feature's area, x-chord, and y-chord. This removed a restriction on the maximum number of features that could be processed. This program also uses the SCALE_FACTOR constant to convert the image pixel size data to values indicating the actual physical size of the particles. This constant value is based on the equipment set up during hologram processing. It is determined by making an image of a known object, such as a threaded screw, at the same magnification. This test image is then used to determine a calibration value for the conversion from pixels to actual feature size.

8. Program File *lstat.c*

The local statistical filter algorithm *lstat.c* (Appendix H) is the first of three filters used in the analysis to reduce speckle noise in the images. This version is a C language version of the FORTRAN program presented by Edwards [Ref. 2]. It uses the disk virtual array functions [Ref. 15] to provide storage for the filtered pixel array during program calculation. The algorithm is described in Reference 16 and basically uses a 5x5 array of pixel values around a central pixel and applies the statistical average obtained from this local array to the central pixel as a new pixel value. Once the calculation for all the pixels in the image is complete, the new pixel values are written to the image screen.

9. Program File *sigma.c*

The program *sigma.c* (Appendix I) is the C version of the 2sigma filter algorithm of Reference 16 and presented by Edwards [Ref. 2]. As with the local statistical filter, this program uses the virtual array functions to store the filtered pixel array during calculation in a disk based virtual array. The algorithm basically takes a 5x5 array average of pixel values and removes all values outside of the 2sigma range. This calculated value is then applied to the 5x5 array central pixel and stored for this central position. Once all pixel positions have been calculated, the new pixel values are written to the image screen.

10. Program File *geofil.c*

The program file *geofil.c* (Appendix J) is the C language adaption of the filter program introduced by Edwards [Ref 2]. This filter uses a geometric hulling algorithm [Ref. 13] to filter the image by comparing pixel values in the horizontal, vertical, and two diagonals around a central pixel. It compares the original image and its complement to determine the proper level to set the central pixel based on the neighboring pixel values. The overall result is the reduction of the speckle noise components at a faster rate than the actual feature pixel data. The complement image array is stored using the virtual disk array routine and thus fully supports

using the entire usable image (512x480). The original program only filtered the first 120 rows of the image (512x120) and provided no means for evaluating any other part of the image.

C. SOFTWARE DEVELOPMENT TOOLS

Several development tools were available for use during program development. Both the MSFORT and MSC compiler packages come with the Micro Soft Codeview Debugger program [Ref. 9 and 12]. This program is an online debugger that allows for program debugging provided special compiler directives are used when compiling the programs.

Since use of Codeview was not easy to accomplish without the extra compiling, a special array *p[x][y]* (Appendix B) was developed to simulate the image screen. This array was then substituted where necessary to simulate use of the library routines that access the image pixels. With the substitution installed, the programs could be compiled and then run to check the results of program execution. A side effect of this process was the ability to also compile the programs using Borland's Turbo C 2.0 [Ref. 17] compiler which includes an **easy to use** online source code debugger within an integrated environment. This process was easier to use than the MSC Codeview program. The TURBO C package is not part of the existing imaging system but should be considered for future upgrades to the system. The package was recently upgraded to version 2.0 and in some respects is better than the MSC package.

Appendix K is a listing and explanation of the DOS Batch files used to compile and link the programs with each compiler. The compiler and linker command directives are those recommended in the ITEX/PC Programmer's guide [Ref. 6]. An explanation of each directive is contained in References 7 and 10 respectively.

III. SOFTWARE PERFORMANCE

This chapter deals specifically with the software performance (both FORTRAN and C) of the programs developed and improved during this research. The improvements in the FORTRAN versions were the reduction of executable program size and the ability to run in the latest DOS environment (DOS Version 4.0). The original programs required the absence of any specially installed device drivers in the DOS environment and would not allow any TSR programs to be loaded during normal processing on the IBM PC/AT. The use of the optimizing compiler allowed all previous programs to execute while coexisting with what is considered a normal set of DOS installed drivers. The *newid* and *newsize* FORTRAN programs, however, still require all of the DOS memory limit for execution and must have any TSR programs removed from memory, but no change in the DOS environment is required before execution.

The improvements using the C language were twofold. The first was a decrease in execution time, and the other was the ability to easily process the entire usable system image size (512x480 pixels) for the first time. The latter seems to be the most significant since future plans to upgrade to the latest ITEX/PC imaging hardware and software will allow an even larger image to be processed.

No specific program testing or verification of the original FORTRAN programs was accomplished. This was deemed unnecessary since no actual coding changes were made in the programs. Only the compiler directives were changed to meet the new format introduced with this version of the compiler.

The testing and verification of the C programs consisted of taking images that were previously processed, recreating a test environment to match that used for the original programs, and then comparing the results against the reported findings of Reference 2 and Reference 3.

A. PROGRAM SIZE INFORMATION

The FORTRAN and C executable program size information is presented in Table 1 under the MSFORT and MSC headings. Each entry is for the EXE (executable file) listing as obtained from the DOS DIR command. The size given is in Kilobytes (KB) for each file as it is stored on disk. The missing table entry for the MSC version of SPECKLE is because this routine is being used in each filter and is included in each C program filter.

The dramatic decrease in size from the original FORTRAN versions to those of the MSFORT compiler are believed to be mostly attributable to the use of the EXEPACK utility of the new compiler [Ref. 7]. The sizes for the MSFORT and MSC versions are relatively consistent with the exception of the filter programs. The added speckle index algorithm and the virtual array code in each filter accounts for the C filters being larger than their FORTRAN counterparts.

TABLE 1
EXECUTABLE PROGRAM SIZE

Filename	Original	MSFORT	MSC
THRESH.EXE	131,882	44,795	31,467
SPECKLE.EXE	122,180	33,569	NONE
NEWSIZE.EXE	556,038	52,575	46,281
NEWID.EXE	569,830	46,655	32,509
SIGMA.EXE	379,704	48,293	49,483
STAT.EXE	544,626	44,143	49,475
GEOFIL.EXE	364,382	41,375	49,969

B. C PROGRAM VERIFICATION AND EXECUTION TIME COMPARISON

The testing and verification of the new C programs consisted of using existing images that were previously analyzed and reported on. The two images chosen are shown in Figure 1 and Figure 2. The image *10xwgrid.img* (Figure 1) was chosen from the image library to test the operation of the *feat_id.c* and *size_it.c* programs. This image shows 23 features that are taken from the standard calibration image used by Redman [Ref. 1]. It displays the 23 different feature sizes that are present in the calibration image. Figure 2 is the *j17res4.img* library image and is an image taken of the AFRT with speckle noise present. This image was used to test each filter program for proper operation. In each test the results obtained were compared against data previously obtained for this project.

The *10xwgrid.img* image was thresholded at a value of 130 and then processed with both the FORTRAN and C versions of the sizing and identification programs. The results of these tests appear to indicate proper operation of the new programs. Both *feat_id.c* and *size_it.c* produced the same outputs as programs *newid.exe* and *newsize.exe* [Ref. 3]. Both counted and sized 23 features. Tables 2 and 3 show the data output for the two programs. The results were normalized to remove the physical size scaling and calculations for conversion to microns. Normalization was required since there does exist a discrepancy between how the programs calculate the final physical size data. The size data obtained from *size_it.c* is about a factor of four smaller than that obtained with the program *newsize*. The SCALE_FACTOR entry in the *thesis.h* header file or the magnification calculation in *newsize* may be incorrect. These values should be adjusted or corrected before a quantitative value for the feature size is used in any critical calculations.

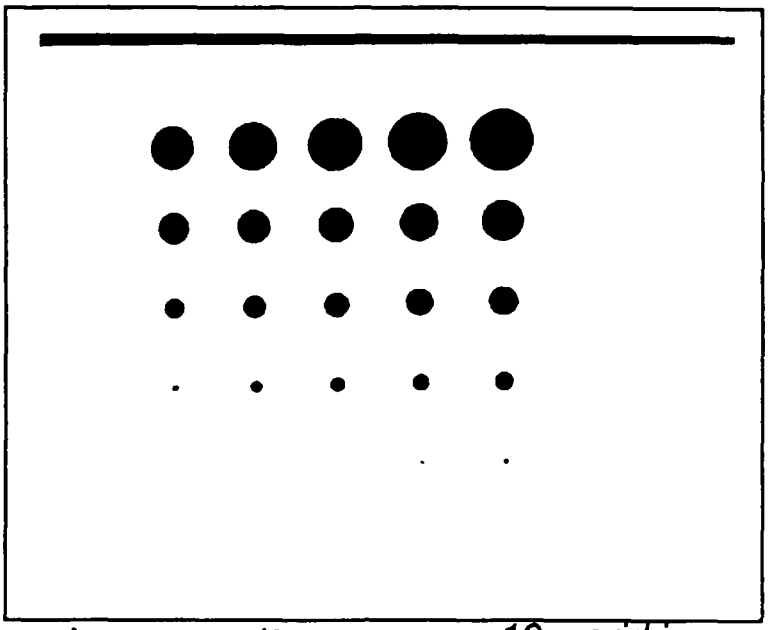

Figure 1 Library Image *10xwgrid.img*

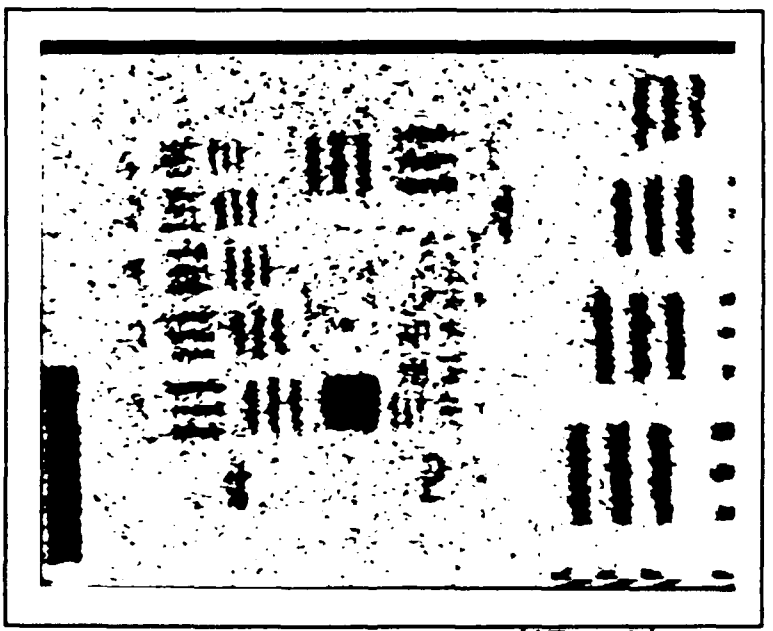

Figure 2 Library Image *j17res4.img*

TABLE 2
PROGRAM *NEWSIZE* OUTPUT DATA.

Num	Area	X-chord	Y-chord
1	101.002	13.000	4.000
2	10.000	5.000	2.000
3	3.000	2.000	1.000
4	6.000	3.000	2.000
5	15.000	4.000	4.000
6	20.000	5.000	6.000
7	70.002	9.000	10.000
8	106.003	11.000	12.000
9	138.003	12.000	14.000
10	177.004	14.000	16.000
11	199.005	15.000	17.000
12	262.006	17.000	20.000
13	310.007	19.000	21.000
14	377.009	21.000	24.000
15	429.010	22.000	25.000
16	478.011	23.000	27.000
17	562.013	25.000	29.000
18	626.015	26.000	30.000
19	733.017	29.000	33.000
20	861.020	31.000	35.000
21	948.023	31.000	39.001
22	1200.029	36.000	43.001
23	1468.035	40.001	46.001

TABLE 3
PROGRAM *SIZE_IT* OUTPUT DATA.

NUM	AREA	X-chord	Y-chord
1	1468.000	40.000	46.000
2	1200.000	36.000	43.000
3	948.000	31.000	39.000
4	861.000	31.000	35.000
5	733.000	29.000	33.000
6	626.000	26.000	30.000
7	562.000	25.000	29.000
8	478.000	23.000	27.000
9	429.000	22.000	25.000
10	377.000	21.000	24.000
11	310.000	19.000	21.000
12	262.000	17.000	20.000
13	199.000	15.000	17.000
14	177.000	14.000	16.000
15	138.000	12.000	14.000
16	106.000	11.000	12.000
17	70.000	9.000	10.000
18	20.000	5.000	6.000
19	15.000	4.000	4.000
20	6.000	3.000	2.000
21	2.000	2.000	1.000
22	10.000	5.000	2.000
23	1.000	1.000	1.000

The *j17res4.img* image was used to test each filter program. The results of the testing are presented in Table 4, Table 5, and Figure 3. Each program was run using a DOS text input file that used DOS redirection to supply the keyboard inputs to the program without operator input. An external software timer measured the times for loading and executing the programs. The timing data in Table 4 were all gathered in this manner. Table 5 is a tabulation of Speckle Index values obtained during each filter operation and contains the data plotted in Figure 3. These data clearly indicate agreement with that achieved previously by Edwards [Ref. 2] and Orguc [Ref. 3].

Figures 4 and 5 show one iteration of the 2sigma filter applied to a full screen image (512x480). The image is once again the *j17res4.img* file that has a pixel value histogram generated by the IMAGEACTION package imbedded into the picture. Comparison of the two histograms visually shows that the filter reduces the speckle content of the image as expected. The presence of the valley and hump to the left of the histogram is indication that the feature grey scale data is emerging above the speckle noise. This result agrees with that achieved by Edwards [Ref. 2].

TABLE 4
PROGRAM EXECUTION TIME COMPARISON

Program	MSFORT	MSC
Feat ID	1 m 11 s	1 m 0 s
Feat Sizing	1 m 15 s	1 m 35 s
2Sigma Filter	7 m 20 s*	6 m 33 s
Local STAT Filter	39 m 45 s*	13 m 0 s
Geometric Filter	18 m 50 s*	6 m 17 s

* [Ref. 2]

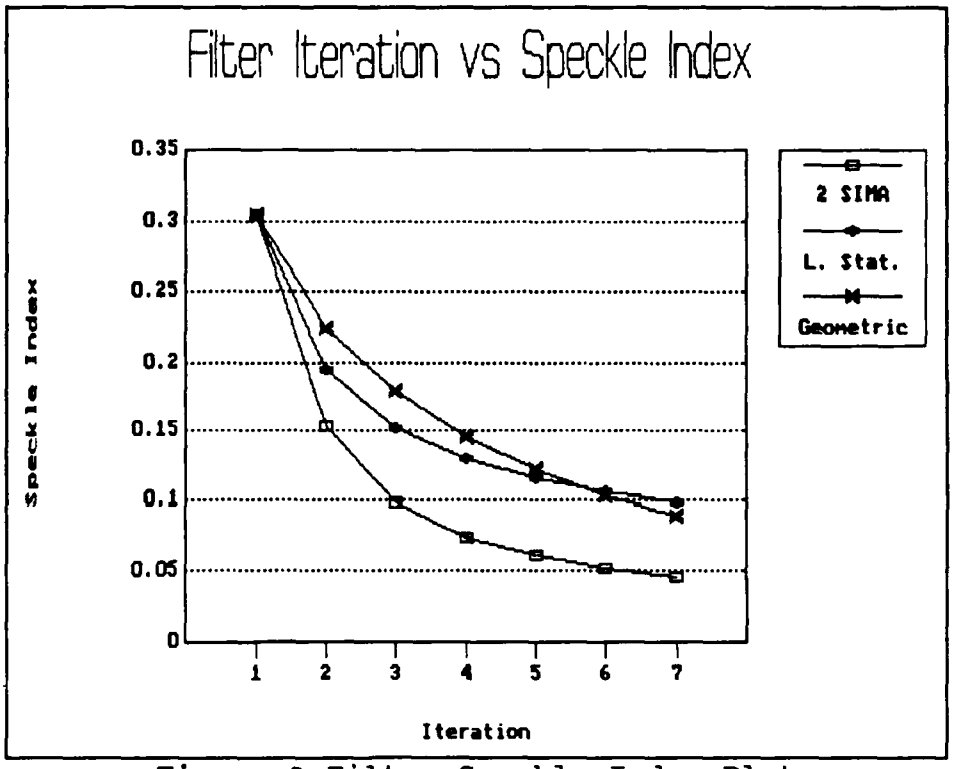

Figure 3 Filter Speckle Index Plot

TABLE 5
SPECKLE INDEX REDUCTION RESULTS

NUM	2 SIGMA	L STAT.	GEOMETRIC
0	0.304005	0.304005	0.304005
1	0.153767	0.194121	0.224875
2	0.098967	0.151951	0.178222
3	0.074415	0.129769	0.146152
4	0.060615	0.115938	0.122347
5	0.051691	0.106372	0.103983
6	0.045354	0.099367	0.089425

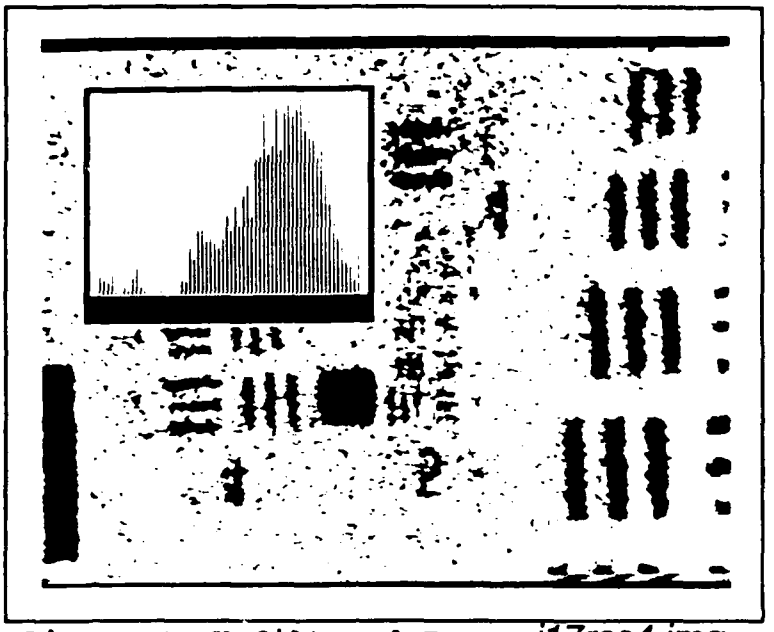

Figure 4 Unfiltered Image *j17res4.img*
(SI = 0.304005)

Figure 5 Filtered Image *j17res4.img*
(SI = 0.153767)

IV. CONCLUSIONS

In summary, this thesis has achieved a significant increase in the ability to perform hologram image analysis on the present imaging system. The goals of smaller compact programs coupled with faster execution times were both achieved. The capability to perform analysis of full 512x480 images in all programs is now available.

Another important aspect is that of portability to the recently acquired upgraded system that includes a COMPAQ 386/20 computer and a new IMAGING Technology Processor and software that only uses the C Language. This new system will be the subject of future research and these programs should port with minimum effort over to the new system. The only requirement should be verification of the required function calls for the library routines used with those of the new library and then the changes made to the header file *thesis.h*. No major program code changes should be required.

It is recommended that further verification testing be conducted with these programs using other images. The testing conducted for this thesis only showed that the programs operate properly and produce results similar to those achieved previously.

APPENDIX A

PROGRAM HEADER FILE: thesis.h

```
/* The purpose of this file is to list all necessary include files,
   manifest constants, and MACRO definitions required by all the
   programs within this package. The programs were written to be as
   portable as possible and, as such, changes to constants in
   this file should be all that is required to change a parameter
   definition throughout the program files.                    */

/*     Include files for use with ITEXPC programs      */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <malloc.h>
#include "itexpc.h"

/*  Initial ITEX/PC Board Jumper Settings   */

#define  MEMBASE    0xD000              /* Itex board base memory start address */
#define  REGBASE    0xFF00             /* Itex board base register start address */
#define  IFLAG      PSEUDO_COLOR             /* Itex board color option installed */

/* ITEXPC initial AOI (area of interest) settings  */

#define IXS         0
#define IYS         0
#define NROW        480
#define NCOL        512

/*  ItexPC LUT (Look Up Table) Variables      */

/*  color board variables   */

#define RED         0
#define GREEN       1
#define BLUE        2
#define INPUT       3

/* Threshold limits  */

#define  LOWEST     0            /* Equates to Black for lowcut value */
#define  HIGHEST    255          /* Equates to White for highcut value */
#define  BLACK      0
#define  WHITE      255

/* Filter array sizes and limits  */

#define  SNUM       25
#define  NUM        9
#define  HIGH       254
#define  LOW        0

/* Sizing magnification factor used in sizing program */

#define  SCALE_FACTOR  3.7353       /* Based on image calibration   */
```

```
/* Virtual array Header File  Definitions   */
#define header 7

/* Virtual Array Control Block typedef  */

typedef struct {

    FILE *file;             /* pointer to file control block */
    long size;               /* number or array elements in file */
    int elsize;             /* number or bytes in each element */
    char *buffer;            /* pointer to array buffer */
    int buf_elsize;         /* size of element in buffer including index */
    int buf_size;           /* number of elements in buffer */
    char *blank_rec;          /* pointer to initialization record */
                            /* used for extending file */
}
VACB ;              /* Virtual Array control block type name */

/* Virtual Array Access Prototypes */

int init_v_array(char *filename,int rec_size,char filchar);
VACB *open_v_array(char *filename,int buffer_size);
void close_v_array(VACB *v_array);
void *access_v_rec(VACB *v_array,long index);

/* Virtual Array Access Macros */

#define VREC(y) ((items *)access_v_rec(item_array,y))
#define gdesc(y) VREC(y)->v_gdesc
#define G(x,y) VREC(y)->v_gdesc[x]

/* Virtual Array element structure typedef */

typedef struct {
    unsigned char v_gdesc[NCOL];
}
items;

/*.....*/
```

APPENDIX B

PROGRAM FILE: genfunc.c

```
/*****************************************************************************
  TITLE    : Image Processing General Support Functions

  CALLED BY: various routines

  FILENAME : genfunc.c

  ____________________________________

  LAST
  MODIFIED : 12/8/1988

  ____________________________________

  PURPOSE  :
            These general functions provide for routine evolutions
          that occur a number of times in the Image processing
          routines developed for analysis of the speckle reduction
          algorithms. Some require support of other functions and
          are not totally independent. This file serves as
          documentation only and has not been nor can it be compiled
          as a stand-alone program file.

*****************************************************************************/

/* initial ITEXPC Board setup  */

startit()
{
    sethdw(REGBASE,MEMBASE,IFLAG);
    aclear(IXS,IYS,NCOL,NROW,150);
    initialize();
    system("cls");       /* Clears monitor screen  */
}

/* Get integer keyboard input  */

geti(name,iptr)
char *name;
int *iptr;
{
    printf(" %s ",name);
    scanf(" %d",iptr);
}
/* Get floating point keyboard input  */

getf(rname,riptr)
char *rname;
int *riptr;
{
    printf("%s ",rname);
    scanf(" %f",riptr);
}
/* Pause routine used to prompt for operator intervention*/
wait()
{
    printf("\n\n Press Return to continue");
```

```
    fflush(stdin);
    getchar();
}
/* returns minimum integer value from input array passed */

getmin(data,num)

int *data;
int num;
{
    int t, min;

    for (min= data[0], t=1 ; t<num ; t++)
        if( data[t] < min ) min = data[t];

    return min;
}

/* returns maximum integer value from input array passed */

getmax(data,num)

int *data;
int num;
{
    int t, max;

    for (max= data[0], t=1 ; t<num ; t++)
        if( data[t] > max ) max = data[t];

    return max;
}

/* DOS System Call to clear Display Screen  */

cls()
{
    system("cls");
}

/*.....*/
/* end of function programs  */



/*.....TESTER.....  ....*/

/*   The following defined array was and can be used to test routines
   before they are applied to the imaging system (i.e., totally offline
   with any computer). This particular array was used in the debugger
   program to verify proper operation of the algorithms before they were
   tested using the full image.  To use the tester, include it in the
   program and change the calls to RPIXEL and WPIXEL as needed.


     Change rpixel(x,y) to p[y][x] and use the following data array

     NOTE: The order of subscript changes '[y][x]' is due to the way C calls
        an array.

     To simulate different screen attributes, change array values as
     needed to simulate the image screen desired.                    */
```

```
static int  p[15][20] = {
{   1,  2,  3,  4,  5,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255},
{   1,  2,  3,  4,  5,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255},
{   1,  2,  3,  4,  5,  0,  3,255,255,255,255,255,255,255,255,255,255,255,255,255},
{   1,  2,  3,  4,  5,  0,  3,255,255,255,255,255,255,255,255,255,255,255,255,255},
{   1,  2,  3,  4,  5,  0,  3,255,255,255,255,255,255,255,255,255,255,255,255,255},
{   1,  2,  3,  4,  5,  0,255,255,255,255,255,  5,  5,255,255,255,  6,  6,  6,  6},
{   0,  0,  0,  0,  0,  0,255,255,255,255,255,  5,  5,  5,  5,255,255,  6,  6,  6,  6},
{ 255,  4,  4,255,255,255,255,255,255,255,  5,  5,  5,  5,  5,  5,255,  6,  6,  6,  6},
{ 255,  4,  4,  4,  4,  4,255,255,  5,  5,  5,  5,  5,  5,  5,  5,  6,  6,  6,  6},
{ 255,255,255,  4,  4,  4,255,255,255,  5,  5,  5,  5,  5,  5,255,255,255,255,255},
{ 255,255,255,  4,  4,255,255,255,255,255,  5,  5,  5,  5,255,255,255,255,255,255},
{ 255,255,  4,  4,  4,  4,255,255,255,255,255,  5,  5,255,255,255,255,255,255,255},
{ 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255},
{ 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  7,  7},
{ 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  7}
};

/*.....................END TESTER.....................................*/
```

APPENDIX C

PROGRAM FILE: threshit.c

```
/*****************************************************************************
  TITLE    : Image THRESHOLD Routine

  FILENAME : threshit.c

  ____________________________________________

  LAST
  MODIFIED : 12/9/88

  ____________________________________________

  PURPOSE  :
             This program thresholds the image on screen by taking
           an operator input value and forcing all image pixel
           values above the threshold value to the value of BLACK
           and all those below the value to the WHITE value.

           ( Background == WHITE ; Feature == BLACK )


*****************************************************************************/
#include "thesis.h"

LS20    filename;     /* LS20 & LS200 are required for ItexPC */
LS200   comline;      /* readim() and saveim() library routines  */

main()
{
    int flag;

    printf("\nThis program will threshold the input image desired.");
    printf("\n\n\tReady to Load Image?...Yes (y) / No (n) ");
    flag=getch();
    if(flag == 'Y'|| flag == 'y')
    {
        startit();
        readit();
    }

    if(flag == 'N'|| flag == 'n')
    {
        initialize();
        exit(0);
    }
    printf("\n\n\tTHRESHOLD the image?...Yes (y) / No (n) ");
    flag = getch();
    if(flag == 'Y'|| flag == 'y') threshit();
}
/*.....*/

threshit()
{
    int  ans, c, option;
    unsigned int val;
```

```
    while(1)
    {
        initialize();
        geti("\n\n\tENTER NEW THRESHOLD LEVEL (0-255) :",&val);
        setlut(INPUT,0);
        threshold(val,HIGHEST);
        setlut(RED,0);
        threshold(val,HIGHEST);
        setlut(GREEN,0);
        threshold(val,HIGHEST);
        setlut(BLUE,0);
        threshold(val,HIGHEST);
        setlut(INPUT,0);

        printf("\n\nSATISFIED WITH THRESHOLD AT %d ?...Yes (y) / No (n)...",val);
        ans = getch();
        if(ans == 'Y' || ans == 'y') break;
    }
    printf("\n\t\t 0: SAVE THE THRESHOLDED IMAGE ?...");
    printf("\n\t\t 1: QUIT (Leave thresholded image for further processing.)...");
    printf("\n\t\t 2: RESTORE System to original input image...");
    geti("\n\t  Select option by NUMBER: ",&option);
    switch(option)
    {
    case 0:
        maplut(IXS,IYS,NCOL,NROW);  /* Map the image to frame memory  */
        saveit();
        break;
    case 1 :
        maplut(IXS,IYS,NCOL,NROW);  /* Map the image to frame memory  */
        break;
    case 2 :
        initialize();    /* initialize without a mapping operation */
        break;
    default:
        break;
    }
}

/*.....*/

startit()                   /* initial ITEXPC Board setup  */
{
    sethdw(REGBASE,MEMBASE,IFLAG);
    aclear(IXS,IYS,NCOL,NROW,150);
    initialize();
    system("cls");              /* Clears monitor screen  */
}
/*.....*/

readit()
{
    int errval;
    int ch;

    while(1)
    {
        printf("\n\nENTER IMAGE FILENAME as (DEV:name.IMG)-->");
        fflush(stdin);
        gets(filename.s);
        filename.l = (char)strlen(filename.s);

        errval = readim(IXS,IYS,NCOL,NROW,filename,&comline);
```

```
        if( errval == 0 ){
            printf("\nFILE-> %s \n\nCOMMENT: %s ",filename.s,comline.s);
            break;
        }
        if( errval != 0 )
        {
            printf("\n\aError reading File!!");
            if( errval == -2 ) printf("\n\tFile Not Found!");
            if( errval == -3) printf("\n\t\tBad File Format!");
            printf("Try Again?? ...Yes (y) / No (n) ");
            ch = getch();
            if( ch ==  'Y' || ch ==  'y' ) continue;
            if( ch ==  'N' || ch ==  'n' ) exit(0);
        }
    }
}

/*.....*/

saveit()
{
    int comflag,errval;
    int ch;

    initialize();

    while(1)
    {
        printf("\n\nENTER FILENAME...( DEV: name.IMG )->");
        fflush(stdin);
        gets(filename.s);
        filename.l = (char)strlen(filename.s);
        geti("\n Enter File Compression value ?...( 0/1 )  ",&comflag);
        printf(" \n\n Enter Image Comment or Return ..to continue.. (Max 200 CHAR):...");
        fflush(stdin);
        gets(comline.s);
        comline.l = (char) strlen(comline.s);

        if (comflag == 1) printf("\n\nStoring Image using Compression -- please wait!!");
        if (comflag == 0) printf("\nStoring Image -- plese wait!! ");

        errval = saveim( IXS,IYS,NCOL,NROW,comflag,filename,comline );

        if( errval == 0)
        {
            printf("\nImage Save completed SAT..");
            break;
        }
        if( errval != 0 )
        {
            printf("\n\a Error saving File!!");
            if( errval == -1 ) printf( "\nInsufficient Disk space" );
            printf("\n\tTry Again?? ...Yes (y) / No (n) ");
            ch = getch();
            if( ch == 'Y' || ch == 'y' ) continue;
            if( ch == 'N' || ch == 'n' ) break;
        }
    }
}

/*.....*/

geti(name,iptr)             /* Read/Get integer keyboard input  */
```

```
char *name;
int *iptr;
{
    printf(" %s :",name);
    scanf(" %d",iptr);
}

/*.....*/
```

APPENDIX D

PROGRAM FILE: speckle.c

```
/*****************************************************************************
  TITLE    : Speckle Noise Reduction Function

  FILENAME : speckle.c

  CALLED BY: All Filter routines

  ------------------------------------------------

  LAST
  MODIFIED : 11/27/1988

  ------------------------------------------------

  PURPOSE  :
              Provides for calculating  the  speckle index of an image.
           The resulting value is used as a measure to evaluate the
           success of filtering an image to reduce the speckle noise.
           This routine is called in all filter algorithms.

       -> OPERATOR must enter a Number of ROWS and COLUMNS
          THE PROGRAM uses to calculate the speckle index.

              ROWS —>  Maximum = 480      Minimum = 1
              COLS —>  Maximum = 512      Minimum = 1

*****************************************************************************/

speckle()
{
    extern int srow, scol;
    int  m, n, x, y, nn;
    long smax, smin;
    long sdata[NUM];

    unsigned long deviation, ssum;
    float  smean, slocal, stotal, tot;
    float  spklindex;

    if( srow >= NROW ) srow = NROW;
    if( scol >= NCOL ) scol = NCOL;

    tot = (long)srow * scol;

    /* Commence calculation   */

    printf("\n\n\tCalculating Speckle INDEX...");

    for ( n = 1 ; n < srow-2 ; n++)
    {
        for ( m = 1 ; m < scol-2 ; m++)
        {
            smax = 0;
            smin = 260;
            ssum = 0;
            nn = 0;
```

```
            for( y = ( n-1 ) ; y < (n+2) ; y++ )
            {
                for( x = ( m-1 ) ; x < ( m+2 ) ; x++)
                {
                    sdata[nn] = rpixel( x,y );
                    ssum += (long)sdata[nn];
                    if( smax < sdata[nn] ) smax = sdata[nn];
                    if( smin > sdata[nn] ) smin = sdata[nn];
                    nn++;
                }
            }
            deviation = smax - smin;
            if( ssum == 0 ) smean = 1;
            smean = ssum / NUM  ;
            slocal = (float)deviation / smean;
            stotal += slocal;
        }
    }
    spklindex = stotal / tot;
    printf("\n\n\tThe Calculated %d by %d speckle index is %f ",srow,scol,spklindex);
}

/*..... END Speckle Function .....*/
```

APPENDIX E

PROGRAM FILE: vir_arry.c

```
/*****************************************************************************
  TITLE    : Virtual Array Support Functions

  FILENAME : vir_arry.c

  ____________________________________________

  LAST
  MODIFIED : 11/27/1988

  ____________________________________________

  PURPOSE  :
             Provides for the setup of a disk drive virtual
           array that can be indexed as if it were in the calling
           program's data storage area.  The routines provide for disk
           file access when required to retrieve data elements from the
           array. This allows the size of a declared array to be limited
           only by the amount of DISK SPACE available. See "thesis.h"
           header file for setup and MACRO routines that support these
           functions.  Also, program lstat.c fully implements the array
           routines.

*****************************************************************************/
#include "thesis.h"

/* Virtual Array Access Routines */

int init_v_array(filename,rec_size,filchar)
char *filename, filchar;
int rec_size;

{
    long size;
    FILE *f;
    f = fopen(filename,"wb");
    if (f != NULL)
    {
        size = 0;
        fwrite(&size,4,1,f);        /* write array size of zero */
        fwrite(&rec_size,2,1,f);    /* and array element size   */
        fwrite(&filchar,1,1,f);     /* and fill char            */
        fclose(f);                  /* to file header */
        return(1);
    }
    else
        return(NULL);
}
VACB *open_v_array(filename,buffer_size)
char *filename;
int buffer_size;
{
    VACB *v_array;
    char *buf_ptr;
    int i;
    char filchar;
```

```
    /* allocate virtual array control block */

    v_array = (VACB *) malloc(sizeof(VACB));
    if (v_array == NULL) return(NULL);

    /* open virtual array file */

    v_array->file = fopen(filename,"r+b");
    if (v_array->file == NULL)
    {
        free(v_array);
        return(NULL);
    };

    /* get array size and element size for control block */

    fread(&v_array->size,4,1,v_array->file);
    fread(&v_array->elsize,2,1,v_array->file);
    fread(&filchar,1,1,v_array->file);
    v_array->buf_elsize = v_array->elsize + 4;

    /* allocate buffer */


    v_array->buffer = (char *) malloc(v_array->buf_elsize * (buffer_size + 1));
    if (v_array->buffer == NULL)
    {
        fclose(v_array->file);
        free(v_array);
        return(NULL);
    };
    v_array->buf_size = buffer_size;

    /* set up blank_rec using the fill character in array header */
    /* for initializing new array elements */

    buf_ptr = v_array->buffer + v_array->buf_elsize * v_array->buf_size;
    v_array->blank_rec = buf_ptr + 4;
    for (i = 0; i < v_array->buf_elsize; i++)
        *buf_ptr++ = filchar;

    /* set record index negative for all buffer elements */

    buf_ptr = v_array->buffer;
    for (i = 0; i < v_array->buf_size; i++)
    {
        *((long *)buf_ptr) = -1L;
        buf_ptr += v_array->buf_elsize;
    };
    return(v_array);
}

void close_v_array(v_array)
VACB *v_array;
{
    int i;
    char *buf_ptr;
    long rec_index, file_offset;

    buf_ptr = v_array->buffer;


    /* flush buffer */
```

```
    for (i = 0; i < v_array->buf_size; i++)
    {

        /* check each element index */
        /* if element present; write it to disk */

        rec_index = *((long *)buf_ptr);
        if (rec_index >= 0)
        {
            file_offset = header + rec_index * v_array->elsize;
            fseek(v_array->file,file_offset,0);
            fwrite(buf_ptr + 4, v_array->elsize,1, v_array->file);
        };
        buf_ptr += v_array->buf_elsize;
    };
    free(v_array->buffer);                  /* de-allocate buffer */
    fclose(v_array->file);                  /* close array file   */
    free(v_array);                          /* de-allocate VACB   */
}

void *access_v_rec(v_array,index)
VACB *v_array;
long index;
{

    char *buf_ptr;
    int buf_index;
    long rec_index, temp_index;


    /* calculate buffer address of referenced element */

    buf_index = index % v_array->buf_size;
    buf_ptr = v_array->buffer + buf_index * v_array->buf_elsize;
    rec_index = *(long *)buf_ptr;

    /* if element present, return its buffer address */
    if (rec_index == index) return(buf_ptr + 4);

    /* if element doesn't exist, extend the file */

    if (index >= v_array->size)
    {
        fseek(v_array->file,0,2);
        for (temp_index = v_array->size; temp_index++ <= index; )
            fwrite(v_array->blank_rec, v_array->elsize, 1, v_array->file);
        v_array->size = index + 1;
        fseek(v_array->file,0,0);
        fwrite(&v_array->size, 4, 1, v_array->file);
    };

    /* if buffer slot is occupied by another element, */
    /* save it to disk  */

    if (rec_index >= 0)
    {
        fseek(v_array->file, rec_index * v_array->elsize + header, 0);
        fwrite(buf_ptr + 4, v_array->elsize, 1, v_array->file);
    };

    /* read referenced element into buffer slot */

    fseek(v_array->file, index * v_array->elsize + header, 0);
```

```
    fread(buf_ptr + 4, v_array->elsize, 1, v_array->file);
    *((long *)buf_ptr) = index;

    /* return address of element in buffer */

    return(buf_ptr + 4);
}

/*.....*/
```

APPENDIX F

PROGRAM FILE: feat_id.c

```
/*****************************************************************************
  TITLE    : Image FEATURE Identification Algorithm

  FILENAME : feat_id.c
  ---------------------------------------------------
  LAST
  MODIFIED : 12/9/1988

  ---------------------------------------------------
  PURPOSE  :
            Labels and identifies each feature in an image on the ITEX
          system.  Reads pixel-by-pixel and groups the objects by
          assigning a unique ID number to each feature or object so
          that they can be processed by other routines.

            The ID routine requires a thresholded image on screen or
            input of a previously saved thresholded image from disk or
            input from disk and then thresholding. This program module
          includes an optional call to threshold if desired.

          ( Background == WHITE ; Feature == BLACK )

*****************************************************************************/
#include "thesis.h"

LS20    filename;     /* LS20 & LS200 are required for ItexPC */
LS200   comline;      /* readim() and saveim() library routines  */


main()
{
    int flag;

    printf("\n\aYou MUST USE A THRESHOLDED IMAGE for this Program!!");
    printf("\n\n\tReady to Load image from disk?...Yes (y) / No (n) ");
    flag=getch();
    if(flag == 'Y'|| flag == 'y')
    {
        startit();
        readit();
    }
    if(flag ==  'N' || flag ==  'n' ) exit(0);

    printf("\n\n\tNeed to THRESHOLD the image?...Yes (y) / No (n) ");
    flag = getch();
    if(flag == 'Y'|| flag == 'y') threshit();

    feat_id();

    printf("\n\nSave image to Disk File?..Yes (y) / No (n) ");
    flag=getch();
    if(flag == 'Y'|| flag == 'y') saveit();
}
```

```
/*.....*/

feat_id()
{
    register int x,y;

    int a,x1,y1,x2,y2,x3,y3,x4,y4;
    int fid = 0;
    int gid = 0;
    int temp_fid = 0;
    int temp_gid = 0;
    int maxfl,n1,n2,n1a,n1b,n1c,kk,kk1,nn1;

    printf("\n\n\tSTEP ONE in Progress...");

    for( y = 1 ; y < NROW ; y++ )
    {
        for( x = 0 ; x < NCOL ; x++ )
        {
            if( rpixel(x,y) == WHITE ) continue;
            if ( x == 0 )
            {
                wpixel( x,y,fid );
                continue;
            }
            if( ( rpixel( x-1,y ) != WHITE ) )
            {
                wpixel( x,y,rpixel(x-1,y) );
                continue;
            }
            if( rpixel(x,y-1) != WHITE )
            {
                wpixel( x,y,rpixel( x,y-1 ) );
                continue;
            }
            wpixel( x,y,fid );
            a = x + 1;

            while(1)
            {
                if( rpixel( a,y ) == WHITE )
                {
                    wpixel( x,y,fid );
                    step( &fid,&gid );
                    break;
                }
                if( rpixel(a,y-1) != WHITE )
                {
                    wpixel( x,y,rpixel( a,y-1 ) );
                    break;
                }
                a += 1;
            }
        } /* end x - for */
    }     /* end y - for */

    fid += gid*HIGH;

    printf("\n\n\tSTART STEP TWO... ");
    printf("\n\nEnter the BEST estimate of the max feature length");
    geti("\n\nIt's Better to be too large than too small!!",&maxfl);
    printf("\n\n\tRUNNING");

    for( y = 1 ; y < NROW ; y++ )
```

```
{
    for( x = 1 ; x < NCOL ; x++ )
    {
        n1 = rpixel( x,y );
        n2 = rpixel( x,y-1 );
        if( ( n1 || n2 == WHITE ) || (n1 == n2 )) continue;

        x3 = x - maxfl;
        x4 = x + maxfl;
        y3 = y - maxfl;
        y4 = y + maxfl;

        if( x3 < LOW) x3 = LOW;
        if( x4 > NCOL ) x4 = NCOL;
        if( y3 < LOW) y3 = LOW;
        if( y4 > NROW ) y4 = NROW;

        for ( y1 = y3 ; y1 < y4 ; y1++ )
        {
            for ( x1 = x3 ; x1 < x4 ; x1++)
            {
                n1a = rpixel( x1,y1 );
                if ( n1a == WHITE) continue;
                if ( n1a == n1 ) wpixel( x1,y1,n2 );
            }
            fid -= 1;
        }
    }
}

printf("\n\n\tSTEP THREE in Progress");

for ( y = 1 ; y < NROW ; y++ )
{
    for( x = 1 ; x < NCOL ; x++ )
    {
        kk = rpixel( x,y );
        if( kk == WHITE ) continue;
        kk1 = kk;
        nn1 = temp_fid + temp_gid *  HIGH;
        if( kk1 < nn1 ) continue;
        if( (kk1 - nn1) < -350 ) kk1 = kk + 508;
        if( (kk1 - nn1) >= -350  && (kk1 - nn1) < -150 )
            kk1 = kk + 254;
        if( kk1 != nn1 )
        {
            x3 = x - maxfl;
            x4 = x + maxfl;
            y4 = y + maxfl;

            if( x3 < 0) x3 = LOW;
            if( x4 > NCOL-1 ) x4 = NCOL -1;
            if( y4 > NROW-1 ) y4 = NROW -1;

            for( y1 = y ; y1 < y4 ; y1++ )
            {
                for( x1 = x3 ; x1 < x4 ; x1++)
                {
                    n1c = rpixel( x1,y1 );
                    if( ( n1c == WHITE ) || ( n1c != kk ) )continue;
                    wpixel( x1,y1,temp_fid  );
                }
            }
            step( &temp_fid,&temp_gid );
```

```
            }
        }
    }

    printf("\n\n\t\t\tFEATURE COUNT IS:   %d",fid);
    maplut( IXS, IYS, NCOL, NROW );
}

step( u,v )             /* counter step routine   */
int *u, *v ;
{
    *u +=1;
    if ( *u >= HIGH )
    {
        *u = 1;
        *v +=1;
    }
}

/*.....*/

threshit()
{
    int  ans, c, option;
    unsigned int val;

    while(1)
    {
        initialize();
        geti("\n\n\tENTER NEW THRESHOLD LEVEL (0-255) :",&val);
        setlut(INPUT,0);
        threshold(val,HIGHEST);
        setlut(RED,0);
        threshold(val,HIGHEST);
        setlut(GREEN,0);
        threshold(val,HIGHEST);
        setlut(BLUE,0);
        threshold(val,HIGHEST);
        setlut(INPUT,0);

        printf("\n\nSATISFIED WITH THRESHOLD AT %d ?...Yes (y) / No (n)...",val);
        ans = getch();
        if(ans == 'Y' || ans == 'y') break;
    }
    maplut(IXS,IYS,NCOL,NROW);
    initialize();
}

/*.....*/

geti(name,iptr)             /* Get integer keyboard input  */
char *name;
int *iptr;
{
    printf(" %s ",name);
    scanf(" %d",iptr);
}

getf(rname,riptr)           /* Get floating point keyboard input  */
char *rname;
int *riptr;
{
    printf("%s ",rname);
    scanf(" %f",riptr);
```

```
}

/*.....*/

startit()                    /* initial ITEXPC Board setup  */
{
    sethdw(REGBASE,MEMBASE,IFLAG);
    aclear(IXS,IYS,NCOL,NROW,150);
    initialize();
    system("cls");               /* Clears monitor screen  */
}
/*.....*/

readit()
{
    int errval;
    int ch;

    while(1)
    {
        printf("\n\n\tENTER IMAGE FILENAME as (DEV:name.IMG)->");
        fflush(stdin);
        gets(filename.s);
        filename.l = (char)strlen(filename.s);

        errval = readim(IXS,IYS,NCOL,NROW,filename,&comline);

        if( errval == 0 ){
            printf("\nFILE-> %s \n\nCOMMENT: %s ",filename.s,comline.s);
            break;
        }
        if( errval != 0 )
        {
            printf("\n\t\aError reading File!!");
            if( errval == -2 ) printf("\n\tFile Not Found!");
            if( errval == -3) printf("\n\t\tBad File Format!");
            printf("\n\tTry Again?? ...Yes (y) / No (n) ");
            ch = getch();
            if( ch ==  'Y' || ch ==  'y' ) continue;
            if( ch ==  'N' || ch ==  'n' ) exit(0);
        }
    }
}
/*.....*/

saveit()
{
    int comflag,errval;
    int ch;

    initialize();

    while(1)
    {
        printf("\n\n\tENTER FILENAME...( DEV: name.IMG )->");
        fflush(stdin);
        gets(filename.s);
        filename.l = (char)strlen(filename.s);
        geti("\n\nEnter File Compression value ?...( 0/1 )  ",&comflag);
        printf("\n\nEnter Image Comment or Return... to continue.. (Max 200 CHAR):...");
        fflush(stdin);
        gets(comline.s);
        comline.l = (char) strlen(comline.s);
```

```
        if (comflag == 1) printf("\n\n\tStoring Image using Compression -- please wait!!");
        if (comflag == 0) printf("\n\n\tStoring Image -- plese wait!! ");

        errval = saveim( IXS,IYS,NCOL,NROW,comflag,filename,comline );

        if( errval == 0)
        {
            printf("\n\nImage Save completed SAT..");
            break;
        }
        if( errval != 0 )
        {
            printf("\n\aError saving File!!");
            if( errval == -1 ) printf( "\nInsufficient Disk space" );
            printf("\n\tTry Again?? ...Yes (y) / No (n) ");
            ch = getch();
            if( ch == 'Y' || ch == 'y' ) continue;
            if( ch == 'N' || ch == 'n' ) break;
        }
    }
}

/*.....*/
```

APPENDIX G

PROGRAM FILE: sizeit.c

```
/*.....
  TITLE    : Image FEATURE SIZING Algorithm

  FILENAME : sizeit.c
  ____________________________________

  LAST
  MODIFIED : 11/26/1988

  ____________________________________

  PURPOSE  :
              Uses an existing or saved image that has been
            thresholded and then ID'd with the ID Algorithm.  The output
            of this program is a tabular output of the calculated Area,
            X-chord, and Y-chord that is suitable for input to a
            Statistical Analysis Program.  Output file is written to
            default drive as SIZE.DAT. This file should be saved under
            different name each time created as subsequent program
            execution will overwrite the file without warning.

            ( Background == WHITE ; Feature == BLACK )

.....*/

#include "thesis.h"

/* Initial Variable Setup/Definitions   */

LS20    filename;     /* LS20 & LS200 are required for ItexPC */
LS200   comline;      /* readim() and saveim() library routines */

main()
{
    int flag;

    printf("\n\aUSE IMAGE processed by the FEATURE ID program!!");

    printf("\n\tReady to Load ID'd image from disk?...Yes (y) / No (n) ");
    flag=getch();
    if(flag == 'Y'|| flag == 'y')
    {
        startit();
        readit();
    }
    if( flag ==  'N' || flag ==  'n' ) exit(0);
    initialize();
    size_it();

    printf("\n\tSave image to Disk File?...Yes (y) / No (n)  ");
    flag=getch();
    if(flag == 'Y'|| flag == 'y') saveit();

}
/*.....*/
```

```
size_it()
{
    register int  nn, x, y, xa, ya, xb, yb;

    int  j, ip, ipa, ipb, x1, y1, x2, y2, maxfl, num;
    int  xmax, ymax, xmin, ymin, maxarea, minarea;

    int  xflag=0;
    int  yflag=0;
    int  aflag=0;

    float  area, xchord, ychord;
    float  fxmax, fymax, fxmin, fymin, fmaxarea, fminarea;
    float  cf_1, cf_2, mag;

    long  *aptr;
    long  *xptr;
    long  *yptr;
    long  *stop;

    FILE  * featdata;

    geti("\nEnter the number of Features",&num);
    getf("\nEnter Magnification Factor",&mag);
    geti("\nEnter the Maximum Feature Length",&maxfl);

    /* Set initial test min/max values  */

    xmax = 0;
    ymax = 0;
    maxarea = 0;
    xmin = 512;
    ymin = 512;
    minarea = 512;

    /* Dynamically allocate memory space for the number of
       features entered by the operator.

          Note: This eliminates the need to declare large arrays
          for these values and allows maximum number of features    */

    xptr = ( long *) calloc( num ,sizeof(long));
    yptr = ( long *) calloc( num ,sizeof(long));
    aptr = ( long *) calloc( num ,sizeof(long));

    /* Check for successfull memory allocation   */

    if( !xptr || !yptr || !aptr )
    {
        printf("\nOut of Memory!!! \n");
        printf("\nYou may have entered too many features.\n");
        printf("\nCorrect problem and try again.");
        exit(0);
    }

    /* Calculate Magnification Constants   */

    cf_1 = ( SCALE_FACTOR / mag );
    cf_2 = ( cf_1 * cf_1 );

    /* Above Conversion factor converts size to microns
          and is related to the input magnification for the
          hologram.  The defined ' SCALE_FACTOR ' constant
          value must be set for the proper value based on the
```

```
        equipment set up during actual image acquisition.
        The constant term SCALE_FACTOR is defined in the
        Header File THESIS.H                                    */

/* Begin Sizing Routine   */

printf("\n\n\tStarting to SIZE!...");

nn = 0;          /* Feature number counter  */

for( y = 0 ; y < NROW ; y++ )
{
    if( nn == num )break;       /* Quit when all features sized */
    for( x = 0 ; x < NCOL ; x++ )
    {
        ip = rpixel(x,y);
        if( (ip == WHITE) || (ip == BLACK) )continue;
        if( nn == num )break;   /* Quit when all features sized */

        x1 = x - maxfl;                 /* Set up sizing Box  */
        y1 = y;
        x2 = x + maxfl;
        y2 = y + maxfl;

        if( x1 < LOW)  x1 = LOW;
        if( y1 < LOW)  y1 = LOW;
        if( x2 > NCOL )  x2 = NCOL;
        if( y2 > NROW )  y2 = NROW;

        for( ya = y1 ; ya < y2 ; ya++ )
        {
            for( xa = x1 ; xa < x2 ; xa++ )
            {
                ipa = rpixel(xa,ya);
                if( ipa != ip )continue;
                aflag++;
                xflag++;
            }
            if( xflag >  *(xptr + nn ) ) *(xptr + nn) = xflag;
            xflag = 0;
        }
        *(aptr + nn) = aflag;
        aflag = 0;

        for( xb = x1 ; xb < x2 ; xb++)
        {
            for( yb = y1 ; yb < y2 ; yb++ )
            {
                ipb = rpixel(xb,yb);
                if( ipb != ip ) continue;
                yflag++;
                wpixel(xb,yb,BLACK);
            }
            if( yflag > *(yptr + nn ) ) *(yptr + nn) = yflag;
            yflag = 0;
        }

        /* Calculate Min/Max values  */

        xmax = max( xmax, *(xptr + nn) );
        ymax = max( ymax, *(yptr + nn) );
        maxarea = max( maxarea, *(aptr + nn) ) ;
        xmin = min( xmin, *(xptr + nn) ) ;
        ymin = min( ymin, *(yptr + nn) ) ;
```

```
            minarea = min( minarea, *(aptr + nn) ) ;

            nn++;               /* increment counter  */
            printf("\n\n\tFeature %d complete...",nn);
        }
    }     /*.... end  initial x/y  for loop ....*/

    /* output data section    */

    printf("\n\a Sending output data to screen and FILE SIZE.DAT !");
    /* wait();   */

    if( ( featdata = fopen("size.dat","w") ) == NULL )
    {
        printf("CANNOT Open output file %s\n", *featdata);
        return(1);
    }

    printf("    ID NO       AREA    X-Width   Y-Width  \n");

    for( j = 0 ; j < num ; j++ )
    {
        fprintf( featdata,"%10d %10.3f %10.3f %10.3f\n", j+1, *(aptr + j) * cf_2,
        *(xptr + j) * cf_1, *(yptr + j) * cf_1 );

        printf("\n%10d %10.3f %10.3f %10.3f ", j+1, *(aptr + j) * cf_2,
        *(xptr + j) * cf_1, *(yptr + j) * cf_1 );
    }

    /* This section can be used to put total information on bottom of data file
       if desired.  File written now to be used with STATGRAPHICS              */

    /*     fprintf(featdata,"\n\n\nMax X-chord= %f    Max Y-chord= %f    Max Area= %f ",
        xmax*cf_1,ymax*cf_1,maxarea*cf_2);
        fprintf(featdata,"\nMin X-chord= %f    Min Y-chord= %f    Min Area= %f ",
        xmin*cf_1,ymin*cf_1,minarea*cf_2);  */

    printf("\n\n\nMax X-chord= %f    Max Y-chord= %f    Max Area= %f ",
    xmax*cf_1,ymax*cf_1,maxarea*cf_2);
    printf("\nMin X-chord= %f    Min Y-chord= %f    Min Area= %f ",
    xmin*cf_1,ymin*cf_1,minarea*cf_2);

    /* Close open file and free allocated memory   */

    fclose(featdata);
    free(xptr);
    free(yptr);
    free(aptr);
}

/*.....*/

geti(name,iptr)             /* Read/Get integer keyboard input  */
char *name;
int *iptr;
{
    printf(" %s :",name);
    scanf(" %d",iptr);
}

getf(rname,riptr)       /* Read/Get real(float) keyboard input  */
char *rname;
int *riptr;
{
```

```
    printf(" %s : (Float) ",rname);
    scanf(" %f",riptr);
}

/*.....*/

wait()      /* Pause routine used to prompt operator */
{
    printf("\n\n Press Return to continue");
    fflush(stdin);
    getchar();
}

/*.....*/

startit()                   /* initial ITEXPC Board setup  */
{
    sethdw(REGBASE,MEMBASE,IFLAG);
    aclear(IXS,IYS,NCOL,NROW,150);
    initialize();
    system("cls");              /* Clears monitor screen  */
}
/*.....*/

readit()
{
    int errval;
    int ch;

    while(1)
    {
        printf("\n\n\tENTER IMAGE FILENAME as (DEV:name.IMG)->");
        fflush(stdin);
        gets(filename.s);
        filename.l = (char)strlen(filename.s);

        errval = readim(IXS,IYS,NCOL,NROW,filename,&comline);

        if( errval == 0 ){
            printf("\nFILE-> %s \n\nCOMMENT: %s ",filename.s,comline.s);
            break;
        }
        if( errval != 0 )
        {
            printf("\n\aError reading File!!");
            if( errval == -2 ) printf("\n\tFile Not Found!");
            if( errval == -3) printf("\n\t\tBad File Format!");
            printf("\n\tTry Again?? ...Yes (y) / No (n) ");
            ch = getch();
            if( ch ==  'Y' || ch ==  'y' ) continue;
            if( ch ==  'N' || ch ==  'n' ) exit(0);
        }
    }
}
/*.....*/

saveit()
{
    int comflag,errval;
    int ch;

    initialize();

    while(1)
```

```
    {
        printf("\n\n\tENTER FILENAME...( DEV: name.IMG )-->");
        fflush(stdin);
        gets(filename.s);
        filename.l = (char)strlen(filename.s);
        geti("\nEnter File Compression value ?...( 0/1 )  ",&comflag);
        printf("\n\n Enter Image Comment or Return ..to continue.. (Max 200 CHAR):...");
        fflush(stdin);
        gets(comline.s);
        comline.l = (char) strlen(comline.s);

        if (comflag == 1) printf("\n\tStoring Image using Compression -- please wait!!");
        if (comflag == 0) printf("\n\tStoring Image -- plese wait!! ");

        errval = saveim( IXS,IYS,NCOL,NROW,comflag,filename,comline );

        if( errval == 0)
        {
            printf("\n\tImage Save completed SAT..");
            break;
        }
        if( errval != 0 )
        {
            printf("\n\aError saving File!!");
            if( errval == -1 ) printf( "\nInsufficient Disk space" );
            printf("\n\tTry Again?? ...Yes (y) / No (n) ");
            ch = getch();
            if( ch == 'Y' || ch == 'y' ) continue;
            if( ch == 'N' || ch == 'n' ) break;
        }
    }
}

/*.....*/
```

APPENDIX H

PROGRAM FILE: lstat.c

```
/*.....
  TITLE    : Local Statistical Filter Algorithm

  FILENAME : lstat.c

  ____________________________________

  LAST
  MODIFIED : 12/9/1988

  ____________________________________

  PURPOSE  :
            Provides for filtering an image using the local statistical
           algorithm.  This program processes the image by using a local
           5x5 array of pixels to calculate a statistical value for the
           central pixel of the local array.

             The area of filtering for the image is controlled by the
           operator by keyboard input of number of ROWS and COLS to
           process.  Program will allow for a Maximum input of 480 rows
           and 512 cols with a minimum of 1 row  and 1 column.  Program
           also requests the image Std. Dev. as calculated by Image Action
           Program.  The resulting filtered image pixel value is stored on
           disk in a virtual array, that is written to the image processing
           screen when all calculations are complete.  DISK SPACE REQUIRED
           for virtual array storage is 250KB minimum available on the disk
           at start of the routine.

           Total calculation time for a 512x480 image is 15.5 min.
.....*/

#include "thesis.h"

LS20    filename;     /* LS20 & LS200 are required for ItexPC */
LS200   comline;      /* readim() and saveim() library routines */

int row, col, srow, scol, times;
float dev;

main()

{
    int flag;
    extern int row, col, times;
    extern float dev;

    printf("\n\tReady to Load IMAGE FILE?...Yes (y) / No (n) ");
    flag=getch();
    if(flag == 'Y' || flag == 'y')
    {
        startit();
        readit();
    }
```

```
    geti("\n\nEnter number of rows (Max=480) to use in SPECKLE CALC : ",&srow);
    geti("\nEnter number of columns (Max=512) to us in SPECKLE CALC :  ",&scol);
    geti("\n\n\nEnter number of filter iterations to do...",&times);
    geti("\nEnter the number of IMAGE rows (MAX = 480) to FILTER : ",&row);
    geti("\nEnter the number of IMAGE cols (MAX = 512) to FILTER : ",&col);
    getf("\n\nEnter the standard deviation for the image: ",&dev);

    do
       {
        speckle();
        lstat();
    }
    while( --times > 0 );
    speckle();

    printf("\nSave image to Disk File?..Yes (y) / No (n) ");
    flag=getch();
    if(flag == 'Y' || flag == 'y') saveit();

}

/*.....*/

lstat()

{
    VACB *item_array;
    long x,y;

    extern int row, col, times;
    extern float dev;

    register int m, n, nn;

    int  pixel;

    float sum1, sum2, svar, lvar, tvar;
    float lmean, lmean2, stddev, stddev2, k;

    int ldata[SNUM];   /* SNUM defined in Header file  */

    /* create a virtual array for the array of filtered
       pixel values the size of the image ( 512x480 )    */

    init_v_array("ITEMS.VAR",sizeof(items),NULL);

    /* open the virtual array, reserve buffer space for 100 elements */

    item_array = open_v_array("ITEMS.VAR",100);

    /* start local statistic filter routine   */

    if( row > NROW ) row = NROW;
    if( col > NCOL ) col = NCOL;

    stddev = dev / 255.0 ;          /* Normalize deviation value */

    stddev2 = stddev * stddev ;

    printf("\n\tFilter Running -> %d  Runs to go!!...",times-1);

    for ( y = 2 ; y < row - 2 ; y++)
    {
        for ( x = 2 ; x < col - 2 ; x++)
```

```
        {
            pixel = rpixel( x,y );
            sum1 = 0;
            sum2 = 0;
            nn = 0;

            for( n = y - 2 ; n < y + 3 ; n++ )
            {
                for( m = x - 2 ; m < x + 3 ; m++ )
                {
                    ldata[nn] = rpixel( m,n );
                    sum1 += ldata[nn];
                    nn++;
                }
            }
            lmean =  sum1 / SNUM  ;
            lmean2 = lmean * lmean ;
            for( nn = 0 ; nn < SNUM ; nn++ )
            {
                svar = (ldata[nn] - lmean ) * ( ldata[nn] - lmean );
                sum2 += svar;
            }
            lvar = sum2 / SNUM ;
            tvar = fabs( ( (lvar + lmean2) / (stddev2 + 1) ) - lmean2 ) ;
            k =  tvar / ( ( stddev2 * lmean2 ) + tvar ) ;
            G( x,y ) = (int) ( lmean + k * ( pixel - lmean ) ) ;
        }
    }

    /* write contents of the G array to image screen */

    printf("\n\a\tWriting filtered image to screen...");

    for( y = 2 ; y < row - 2 ; y++ )
    {
        for( x = 2 ; x < col - 2 ; x++ )
            wpixel( x,y,G(x,y) );
    }

    /* close the virtual array */

    close_v_array(item_array);
}

/*.....*/

geti(name,iptr)             /* Get integer keyboard input  */
char *name;
int *iptr;
{
    printf(" %s ",name);
    scanf(" %d",iptr);
}

getf(rname,riptr)           /* Get floating point keyboard input  */
char *rname;
int *riptr;
{
    printf("%s ",rname);
    scanf(" %f",riptr);
}

/*.....*/
```

```
startit()                   /* initial ITEXPC Board setup  */
{
    sethdw(REGBASE,MEMBASE,IFLAG);
    aclear(IXS,IYS,NCOL,NROW,150);
    initialize();
    system("cls");              /* Clears monitor screen  */
}
/*.....*/

readit()
{
    int errval;
    int ch;

    while(1)
    {
        printf("\n\n ENTER IMAGE FILENAME as (DEV:name.IMG)->");
        fflush(stdin);
        gets(filename.s);
        filename.l = (char)strlen(filename.s);

        errval = readim(IXS,IYS,NCOL,NROW,filename,&comline);

        if( errval == 0 ){
            printf("\nFILE-> %s \n\nCOMMENT: %s ",filename.s,comline.s);
            break;
        }
        if( errval != 0 )
        {
            printf("\n\aError reading File!!");
            if( errval == -2 ) printf("\n\tFile Not Found!");
            if( errval == -3) printf("\n\t\tBad File Format!");
            printf("Try Again?? ...Yes (y) / No (n) ");
            ch = getch();
            if( ch ==  'Y' || ch ==  'y' ) continue;
            if( ch ==  'N' || ch ==  'n' ) exit(0);
        }
    }
}

/*.....*/

saveit()
{
    int comflag,errval;
    int ch;

    initialize();

    while(1)
    {
        printf("\n\nENTER FILENAME...( DEV: name.IMG )->");
        fflush(stdin);
        gets(filename.s);
        filename.l = (char)strlen(filename.s);
        geti("\nEnter File Compression value ?...( 0/1 )  ",&comflag);
        printf("\n\n Enter Image Comment or Return ..to continue.. (Max 200 CHAR):...");
        fflush(stdin);
        gets(comline.s);
        comline.l = (char) strlen(comline.s);

        if (comflag == 1) printf("\n\nStoring Image using Compression -- please wait!!");
        if (comflag == 0) printf("\nStoring Image -- plese wait!! ");
```

```
        errval = saveim( IXS,IYS,NCOL,NROW,comflag,filename,comline );

        if( errval == 0)
        {
            printf("\nImage Save completed SAT..");
            break;
        }
        if( errval != 0 )
        {
            printf("\n\aError saving File!!");
            if( errval == -1 ) printf( "\nInsufficient Disk space" );
            printf("\n\tTry Again?? ...Yes (y) / No (n) ");
            ch = getch();
            if( ch == 'Y' || ch == 'y' ) continue;
            if( ch == 'N' || ch == 'n' ) break;
        }
    }
}

/*.....*/

speckle()
{
    extern int srow, scol;
    int  m, n, x, y, nn;
    long smax, smin;
    long sdata[NUM];

    unsigned long deviation, ssum;

    float  smean, slocal, stotal, tot;
    float  spklindex;

    if( srow >= NROW ) srow = NROW;
    if( scol >= NCOL ) scol = NCOL;

    tot = (long)srow * scol;

    /* Commence calculation   */

    printf("\n\n\tCalculating Speckle INDEX...");

    for ( n = 1 ; n < srow-2 ; n++)
    {
        for ( m = 1 ; m < scol-2 ; m++)
        {
            smax = 0;
            smin = 260;
            ssum = 0,
            nn = 0;

            for( y = ( n-1 ) ; y < (n+2) ; y++ )
            {
                for( x = ( m-1 ) ; x < ( m+2 ) ; x++)
                {
                    sdata[nn] = rpixel( x,y );
                    ssum += (long)sdata[nn];
                    if( smax < sdata[nn] ) smax = sdata[nn];
                    if( smin > sdata[nn] ) smin = sdata[nn];
                    nn++;
                }
            }
            deviation = smax - smin;
            if( ssum == 0 ) smean = 1;
```

```
                smean = ssum / NUM  ;
                slocal = (float)deviation / smean;
                stotal += slocal;
            }
        }
        spklindex = stotal / tot;
        printf("\n\n\tThe Calculated %d by %d speckle index is %f ",srow,scol,spklindex);
}

/*.....*/

/* Virtual Array Access Routines */

int init_v_array(filename,rec_size,filchar)
char *filename, filchar;
int rec_size;
{
    long size;
    FILE *f;
    f = fopen(filename,"wb");
    if (f != NULL)
    {
        size = 0;
        fwrite(&size,4,1,f);          /* write array size of zero */
        fwrite(&rec_size,2,1,f);      /* and array element size   */
        fwrite(&filchar,1,1,f);       /* and fill char            */
        fclose(f);                    /* to file header           */
        return(1);
    }
    else
        return(NULL);
}

VACB *open_v_array(filename,buffer_size)
char *filename;
int buffer_size;
{
    VACB *v_array;
    char *buf_ptr;
    int i;
    char filchar;

    /* allocate virtual array control block */

    v_array = (VACB *) malloc(sizeof(VACB));
    if (v_array == NULL) return(NULL);

    /* open virtual array file */

    v_array->file = fopen(filename,"r+b");
    if (v_array->file == NULL)
    {
        free(v_array);
        return(NULL);
    };

    /* get array size and element size for control block */

    fread(&v_array->size,4,1,v_array->file);
    fread(&v_array->elsize,2,1,v_array->file);
    fread(&filchar,1,1,v_array->file);
    v_array->buf_elsize = v_array->elsize + 4;

    /* allocate buffer */
```

```
    v_array->buffer = (char *) malloc(v_array->buf_elsize * (buffer_size + 1));
    if (v_array->buffer == NULL) {
        fclose(v_array->file);
        free(v_array);
        return(NULL);
    };
    v_array->buf_size = buffer_size;

    /* set up blank_rec using the fill character in array header */
    /* for initializing new array elements */

    buf_ptr = v_array->buffer + v_array->buf_elsize * v_array->buf_size;
    v_array->blank_rec = buf_ptr + 4;
    for (i = 0; i < v_array->buf_elsize; i++)
        *buf_ptr++ = filchar;

    /* set record index negative for all buffer elements */

    buf_ptr = v_array->buffer;
    for (i = 0; i < v_array->buf_size; i++)
    {
        *((long *)buf_ptr) = -1L;
        buf_ptr += v_array->buf_elsize;
    };
    return(v_array);
}

void close_v_array(v_array)
VACB *v_array;
{
    int i;
    char *buf_ptr;
    long rec_index, file_offset;

    buf_ptr = v_array->buffer;

    /* flush buffer */

    for (i = 0; i < v_array->buf_size; i++)
    {
        /* check each element index */
        /* if element present; write it to disk */

        rec_index = *((long *)buf_ptr);
        if (rec_index >= 0)
        {
            file_offset = header + rec_index * v_array->elsize;
            fseek(v_array->file,file_offset,0);
            fwrite(buf_ptr + 4, v_array->elsize,1, v_array->file);
        };
        buf_ptr += v_array->buf_elsize;
    };
    free(v_array->buffer);              /* de-allocate buffer */
    fclose(v_array->file);              /* close array file   */
    free(v_array);                      /* de-allocate VACB   */
}

void *access_v_rec(v_array,index)
VACB *v_array;
long index;
{
    char *buf_ptr;
    int buf_index;
```

```
    long rec_index, temp_index;


    /* calculate buffer address of referenced element */

    buf_index = index % v_array->buf_size;
    buf_ptr = v_array->buffer + buf_index * v_array->buf_elsize;
    rec_index = *(long *)buf_ptr;

    /* if element present, return its buffer address */
    if (rec_index == index) return(buf_ptr + 4);

    /* if element doesn't exist, extend the file */

    if (index >= v_array->size)
    {
        fseek(v_array->file,0,2);
        for (temp_index = v_array->size; temp_index++ <= index; )
            fwrite(v_array->blank_rec, v_array->elsize, 1, v_array->file);
        v_array->size = index + 1;
        fseek(v_array->file,0,0);
        fwrite(&v_array->size, 4, 1, v_array->file);
    };

    /* if buffer slot is occupied by another element, */
    /* save it to disk  */

    if (rec_index >= 0)
    {
        fseek(v_array->file, rec_index * v_array->elsize + header, 0);
        fwrite(buf_ptr + 4, v_array->elsize, 1, v_array->file);
    };

    /* read referenced element into buffer slot */

    fseek(v_array->file, index * v_array->elsize + header, 0);
    fread(buf_ptr + 4, v_array->elsize, 1, v_array->file);
    *((long *)buf_ptr) = index;

    /* return address of element in buffer */

    return(buf_ptr + 4);
}

/*.....*/
```

APPENDIX I

PROGRAM FILE: 2sigma.c

```
/*.....
  TITLE    : 2SIGMA Statistical Filter Algorithm

  FILENAME : 2sigma.c

  ---------------------------------------------
  LAST
  MODIFIED : 12/9/1988

  ---------------------------------------------
  PURPOSE  :
             Provides for filtering an image using a 2sigma statistical
           algorithm. This program processes the image by using a local
           5x5 array of pixels to calculate a statistical value for the
           central pixel of the local array. The algorithm will eliminate
           pixels from the summation that are greater than 2sigma in
           value from that of the central pixel.

             The area of filtering for the image is controlled by the
           operator by keyboard input of number of ROWS and COLS to
           process. Program will allow for a maximum input of 480 rows
           and 512 cols with a minimum of 1 row  and 1 column. Program
           also requests the image standard deviation as obtained using
           Image Action Program. The resulting filtered pixel value is
           stored on disk in a virtual array and written to screen later.

           Total calculation time for a 512x480 image is 5.7 min.

           DISK SPACE REQUIRED for virtual array storage is 250 KB
           minimum available on the disk at start of the routine.

.....*/
#include "thesis.h"

LS20    filename;     /* LS20 & LS200 are required for ItexPC */
LS200   comline;      /* readim() and saveim() library routines */

int row, col, srow, scol, times;
float dev;

main()

{
    int flag;
    extern int row, col, times;
    extern float dev;

    printf("\n\tReady to Load IMAGE FILE from disk?...Yes (y) / No (n) ");
    flag=getch();
    if(flag == 'Y' || flag == 'y')
    {
        startit();
        readit();
    }
```

```
    geti("\n\nEnter number of rows (Max=480) to use in SPECKLE CALC : ",&srow);
    geti("\nEnter number of columns (Max=512) to us in SPECKLE CALC :  ",&scol);
    geti("\n\n\nEnter number of filter iterations to do...",&times);
    geti("\nEnter the number of IMAGE rows (MAX = 480) to FILTER : ",&row);
    geti("\nEnter the number of IMAGE cols (MAX = 512) to FILTER : ",&col);
    getf("\n\nEnter the standard deviation for the image: ",&dev);
    do
       {
        speckle();
        sigma();
    }
    while( --times > 0 );
    speckle();

    printf("\nSave image to Disk File?..Yes (y) / No (n) ");
    flag=getch();
    if(flag == 'Y'|| flag == 'y') saveit();

}

/*.....*/

sigma()

{
    VACB *item_array;
    long x,y;

    extern int row, col, times;
    extern float dev;

    register int m, n, nn;

    int  pixel, sum1, sum2, delta, high, low;

    float stddev, hvar, lvar;

    int ldata[SNUM];          /* SNUM defined in Header file  */

    /* create a virtual array for the array of filtered
       pixel values the size of the image ( 512x480 )        */

    init_v_array("ITEMS.VAR",sizeof(items),NULL);

    /* open the virtual array, reserve buffer space for 100 elements */

    item_array = open_v_array("ITEMS.VAR",100);

    /* start local statistic filter routine   */

    if( row > NROW ) row = NROW;
    if( col > NCOL ) col = NCOL;

    stddev = dev / 255.0 ;          /* Normalize deviation value */
    hvar = 1. + 2. * stddev ;       /* find high and low sigma  */
    lvar = 1.  2. * stddev ;

    printf("\n\n\tFilter Running -> %d Runs to go!!...",times-1);

    for ( y = 2 ; y < row - 2 ; y++)
    {
        for ( x = 2 ; x < col - 2 ; x++)
        {
```

```
            pixel = rpixel( x,y );
            sum1 = 0;
            sum2 = 0;
            nn = 0;
            delta = 0;
            high = (int)(hvar * pixel);
            low = (int)(lvar * pixel);

            for( n = y - 2 ; n < y + 3 ; n++ )
            {
                for( m = x - 2 ; m < x + 3 ; m++ )
                {
                    ldata[nn] = rpixel( m,n );
                    if( ( low <= ldata[nn] ) && ( ldata[nn] <= high ) )
                    {
                        sum1 += ldata[nn];
                        delta++;
                    }
                    nn++;
                }
            }

            if(delta <= 2)    /* correct shot noise -- 4 neighbor average */
            {
                sum2 = ( rpixel(x,y-1)+rpixel(x,y+1)+rpixel(x-1,y)+rpixel(x+1,y) );
                G( x,y ) = (pixel + sum2) / 5;
                continue;
            }

            G( x,y ) = sum1 / delta ;
        }
    }

    printf("\n\aWriting filtered image to screen...");

    for( y = 2 ; y < row - 2 ; y++ )
    {
        for( x = 2 ; x < col - 2 ; x++ )
            wpixel( x,y,G(x,y) );
    }

    /* close the virtual array */

    close_v_array(item_array);
}

/*.....*/

geti(name,iptr)          /* Get integer keyboard input  */
char *name;
int *iptr;
{
    printf(" %s ",name);
    scanf(" %d",iptr);
}

getf(rname,riptr)        /* Get floating point keyboard input  */
char *rname;
int *riptr;
{
    printf("%s ",rname);
    scanf(" %f",riptr);
}
```

```
/*.....*/

startit()                    /* initial ITEXPC Board setup  */
{
    sethdw(REGBASE,MEMBASE,IFLAG);
    aclear(IXS,IYS,NCOL,NROW,150);
    initialize();
    system("cls");          /* Clears monitor screen  */
}
/*.....*/

readit()
{
    int errval;
    int ch;

    while(1)
    {
        printf("\n\nENTER IMAGE FILENAME as (DEV:name.IMG)->");
        fflush(stdin);
        gets(filename.s);
        filename.l = (char)strlen(filename.s);

        errval = readim(IXS,IYS,NCOL,NROW,filename,&comline);

        if( errval == 0 ){
            printf("\nFILE-> %s \n\nCOMMENT: %s ",filename.s,comline.s);
            break;
        }
        if( errval != 0 )
        {
            printf("\n\aError reading File!!");
            if( errval == -2 ) printf("\n\tFile Not Found!");
            if( errval == -3) printf("\n\t\tBad File Format!");
            printf("Try Again?? ...Yes (y) / No (n) ");
            ch = getch();
            if( ch ==  'Y' || ch ==  'y' ) continue;
            if( ch ==  'N' || ch ==  'n' ) exit(0);
        }
    }
}
/*.....*/

saveit()
{
    int comflag,errval;
    int ch;

    initialize();

    while(1)
    {
        printf("\n\nENTER FILENAME...( DEV: name.IMG )->");
        fflush(stdin);
        gets(filename.s);
        filename.l = (char)strlen(filename.s);
        geti("\nEnter File Compression value ?...( 0/1 )  ",&comflag);
        printf(" \n\n Enter Image Comment or Return ..to continue.. (Max 200 CHAR):...");
        fflush(stdin);
        gets(comline.s);
        comline.l = (char) strlen(comline.s);

        if (comflag == 1) printf("\n\nStoring Imag  sing Compression - please wait!!");
        if (comflag == 0) printf("\nStoring Image - ease wait!! ");
```

```
        errval = saveim( IXS,IYS,NCOL,NROW,comflag,filename,comline );

        if( errval == 0)
        {
            printf("\nImage Save completed SAT..");
            break;
        }
        if( errval != 0 )
        {
            printf("\n\a Error saving File!!");
            if( errval == -1 ) printf( "\nInsufficient Disk space" );
            printf("\n\tTry Again?? ...Yes (y) / No (n) ");
            ch = getch();
            if( ch == 'Y' || ch == 'y' ) continue;
            if( ch == 'N' || ch == 'n' ) break;
        }
    }
}

/*.....*/

speckle()
{
    extern int srow, scol;
    int  m, n, x, y, nn;
    long smax, smin;
    long sdata[NUM];

    unsigned long deviation, ssum;

    float  smean, slocal, stotal, tot;
    float  spklindex;

    if( srow >= NROW ) srow = NROW;
    if( scol >= NCOL ) scol = NCOL;

    tot = (long)srow * scol;

    /* Commence calculation   */

    printf("\n\n\tCalculating Speckle INDEX...");

    for ( n = 1 ; n < srow-2 ; n++)
    {
        for ( m = 1 ; m < scol-2 ; m++)
        {
            smax = 0;
            smin = 260;
            ssum = 0;
            nn = 0;

            for( y = ( n-1 ) ; y < (n+2) ; y++ )
            {
                for( x = ( m-1 ) ; x < ( m+2 ) ; x++)
                {
                    sdata[nn] = rpixel( x,y );
                    ssum += (long)sdata[nn];
                    if( smax < sdata[nn] ) smax = sdata[nn];
                    if( smin > sdata[nn] ) smin = sdata[nn];
                    nn++;
                }
            }
            deviation = smax - smin;
```

```
            if( ssum == 0 ) smean = 1;
            smean = ssum / NUM  ;
            slocal = (float)deviation / smean;
            stotal += slocal;
        }
    }
    spklindex = stotal / tot;
    printf("\n\n\tThe Calculated %d by %d speckle index is %f ",srow,scol,spklindex);
}

/*.....*/
/* Virtual Array Access Routines */

int init_v_array(filename,rec_size,filchar)
char *filename, filchar;
int rec_size;
{
    long size;
    FILE *f;
    f = fopen(filename,"wb");
    if (f != NULL)
    {
        size = 0;
        fwrite(&size,4,1,f);        /* write array size of zero */
        fwrite(&rec_size,2,1,f);    /* and array element size   */
        fwrite(&filchar,1,1,f);     /* and fill char            */
        fclose(f);                  /* to file header */
        return(1);
    }
    else
        return(NULL);
}

VACB *open_v_array(filename,buffer_size)
char *filename;
int buffer_size;
{
    VACB *v_array;
    char *buf_ptr;
    int i;
    char filchar;

    /* allocate virtual array control block */

    v_array = (VACB *) malloc(sizeof(VACB));
    if (v_array == NULL) return(NULL);

    /* open virtual array file */

    v_array->file = fopen(filename,"r+b");
    if (v_array->file == NULL)
    {
        free(v_array);
        return(NULL);
    };

    /* get array size and element size for control block */

    fread(&v_array->size,4,1,v_array->file);
    fread(&v_array->elsize,2,1,v_array->file);
    fread(&filchar,1,1,v_array->file);
    v_array->buf_elsize = v_array->elsize + 4;

    /* allocate buffer */
```

```
    v_array->buffer = (char *) malloc(v_array->buf_elsize * (buffer_size + 1));
    if (v_array->buffer == NULL) {
        fclose(v_array->file);
        free(v_array);
        return(NULL);
    };
    v_array->buf_size = buffer_size;

    /* set up blank_rec using the fill character in array header */
    /* for initializing new array elements */

    buf_ptr = v_array->buffer + v_array->buf_elsize * v_array->buf_size;
    v_array->blank_rec = buf_ptr + 4;
    for (i = 0; i < v_array->buf_elsize; i++)
        *buf_ptr++ = filchar;

    /* set record index negative for all buffer elements */

    buf_ptr = v_array->buffer;
    for (i = 0; i < v_array->buf_size; i++)
    {
        *((long *)buf_ptr) = -1L;
        buf_ptr += v_array->buf_elsize;
    };
    return(v_array);
}

void close_v_array(v_array)
VACB *v_array;
{
    int i;
    char *buf_ptr;
    long rec_index, file_offset;

    buf_ptr = v_array->buffer;

    /* flush buffer */

    for (i = 0; i < v_array->buf_size; i++)
    {
        /* check each element index */
        /* if element present; write it to disk */

        rec_index = *((long *)buf_ptr);
        if (rec_index >= 0)
        {
            file_offset = header + rec_index * v_array->elsize;
            fseek(v_array->file,file_offset,0);
            fwrite(buf_ptr + 4, v_array->elsize,1, v_array->file);
        };
        buf_ptr += v_array->buf_elsize;
    };
    free(v_array->buffer);              /* de-allocate buffer */
    fclose(v_array->file);              /* close array file   */
    free(v_array);                      /* de-allocate VACB   */
}

void *access_v_rec(v_array,index)
VACB *v_array;
long index;
{
    char *buf_ptr;
    int buf_index;
```

```
    long rec_index, temp_index;


    /* calculate buffer address of referenced element */

    buf_index = index % v_array->buf_size;
    buf_ptr = v_array->buffer + buf_index * v_array->buf_elsize;
    rec_index = *(long *)buf_ptr;

    /* if element present, return its buffer address */
    if (rec_index == index) return(buf_ptr + 4);

    /* if element doesn't exist, extend the file */

    if (index >= v_array->size)
    {
        fseek(v_array->file,0,2);
        for (temp_index = v_array->size; temp_index++ <= index; )
            fwrite(v_array->blank_rec, v_array->elsize, 1, v_array->file);
        v_array->size = index + 1;
        fseek(v_array->file,0,0);
        fwrite(&v_array->size, 4, 1, v_array->file);
    };

    /* if buffer slot is occupied by another element, */
    /* save it to disk  */

    if (rec_index >= 0)
    {
        fseek(v_array->file, rec_index * v_array->elsize + header, 0);
        fwrite(buf_ptr + 4, v_array->elsize, 1, v_array->file);
    };

    /* read referenced element into buffer slot */

    fseek(v_array->file, index * v_array->elsize + header, 0);
    fread(buf_ptr + 4, v_array->elsize, 1, v_array->file);
    *((long *)buf_ptr) = index;

    /* return address of element in buffer */

    return(buf_ptr + 4);
}
/*.....*/
```

APPENDIX J

PROGRAM FILE: geofil.c

```
/*.....
  TITLE    : Geometric Filter Algorithm

  FILENAME : geofil.c

  ---------------------------------------------------
  LAST
  MODIFIED : 12/9/1988

  ---------------------------------------------------
  PURPOSE  :
            Provides for filtering an image using the geometric filter
           algorithm.  This program processes the image and provides
           for the operator to select the number of iterations.

             The area of filtering for the image is controlled by the
           operator by keyboard input of number of ROWS and COLS to
           process.  Program will allow for a Maximum input of 480 rows
           and 512 cols with a minimum of 1 row  and 1 column.

             The resulting filtered image pixel value is stored on
           disk in a virtual array that is written to the image
           processing screen when all calculations are complete.

           Total calculation time for a 512x480 image is 41.7 min.

           DISK SPACE REQUIRED for virtual array storage is 250 KB minimum
           available on the disk at start of the routine.

.....*/

#include "thesis.h"

LS20    filename;     /* LS20 & LS200 are required for ltexPC */
LS200   comline;      /* readim() and saveim() library routines */

int row, col, srow, scol, times;

main()

{
    int flag;
    extern int row, col, srow, scol, times;

    printf("\n\tReady to Load IMAGE FILE?...Yes (y) / No (n) ");
    flag=getch();
    if(flag == 'Y'|| flag == 'y')
    {
        startit();
        readit();
    }
    if( flag ==  'N' || flag ==  'n' ) exit(0);

    geti("\n\nEnter number of rows (Max=480) to use in SPECKLE CALC : ",&srow);
    geti("\nEnter number of columns (Max=512) to us in SPECKLE CALC :  ",&scol);
```

```
    geti("\n\n\nEnter number of filter iterations to do...",&times);
    geti("\nEnter the number of IMAGE rows (MAX = 480) to FILTER : ",&row);
    geti("\nEnter the number of IMAGE cols (MAX = 512) to FILTER : ",&col);

    do
       {
        speckle();
        geofil();
    }
    while( --times > 0 );
    speckle();

    printf("\nSave image to Disk File?..Yes (y) / No (n) ");
    flag=getch();
    if(flag == 'Y' || flag == 'y') saveit();

}

/*.....*/

geofil()
{
    VACB *item_array;
    long x,y;

    extern int row, col, times;

    int  f1, f2, g1, g2, g3;
    int  pixel, a, b, c, d;

    /* create a virtual array for the array of filtered
       pixel values the size of the image ( 512x480 )    */

    init_v_array("ITEMS.VAR",sizeof(items),NULL);

    /* open the virtual array, reserve buffer space for 100 elements */

    item_array = open_v_array("ITEMS.VAR",100);

    /* start local statistic filter routine   */

    if( row > NROW ) row = NROW;
    if( col > NCOL ) col = NCOL;

    for( y = 1 ; y < row ; y++ )      /* Zero the initial array  */
    {
      for( x = 1 ; x < col ; x++ )
        {
          G( x,y ) = NULL;
        }
    }

    printf("\n\n\tFilter Running -> %d Runs after this one!...\n",times-1);

    a=1; b=0; c=3; d=1;
    printf("\nc  = 3 ");

t1: for( y = 1 ; y < row-1 ; y++ )
    {
      for( x = 1 ; x < col-1 ; x++ )
        {
          f1 = rpixel( x,y );
          f2 = rpixel( x-a,y-b );
            G( x,y ) = max( f1, min( f2-1, f1+1 ) );
```

```
        }
    }
    printf(" step 1");
    for( y = 1 ; y < row-1 ; y++ )
    {
     for( x = 1 ; x < col-1 ; x++ )
        {
          g1 =  G( x,y);
            g2 =  G( x-a,y-b );
          g3 =  G( x+a,y+b );
            pixel = max( g1, min( g2, min( g1+1, g3+1 ) ) );
            wpixel( x,y,pixel);
        }
    }
    printf(" step 2");
    if( d==1 )
    {
        a= -a;
        b= -b;
        d=0;
        goto t1;
    }
    if( d==0 )
    {
        d=1;
      goto t2;
     }

t2: for( y = 1 ; y < row-1 ; y++ )
    {
     for( x = 1 ; x < col-1 ; x++ )
        {
            f1 = rpixel( x,y );
          f2 = rpixel( x-a,y-b );
          G( x,y ) = min( f1, max( f2-1, f1+1 ) );
        }
    }
    printf(" step 3");
    for( y = 1 ; y < row-1 ; y++ )
    {
     for( x = 1 ; x < col-1 ; x++ )
        {
          g1 =  G( x,y);
            g2 =  G( x-a,y-b );
          g3 =  G( x+a,y+b );
            pixel = max( g1, min( g2, min( g1+1, g3+1 ) ) );
            wpixel( x,y,pixel);
        }
    }
    printf(" step 4");
    if( d==1 )
    {
        a= -a;
        b= -b;
        d=0;
        goto t2;
    }
    if( d==0 )
    {
     d=1;
     goto t3;
    }

t3: switch(c)
```

```
    {
    case  0: printf("\nc  = 0  END ");
          break;

    case  3: a=0;
             b=1;
             c=2;
             printf("\nc  = 3 ");
             goto t1;

    case  2: a=1;
             b=1;
             c=1;
             printf("\nc  = 2 ");
             goto t1;


    case  1: a=1;
             b= -1;
             c=0;
             printf("\nc  = 1 ");
             goto t1;

    default  : break;

    }

    /* close the virtual array */

    close_v_array(item_array);
}

/*.....*/

geti(name,iptr)              /* Get integer keyboard input  */
char *name;
int *iptr;
{
    printf(" %s ",name);
    scanf(" %d",iptr);
}

getf(rname,riptr)            /* Get floating point keyboard input  */
char *rname;
int *riptr;
{
    printf("%s ",rname);
    scanf(" %f",riptr);
}

/*.....*/

startit()                    /* initial ITEXPC Board setup  */
{
    sethdw(REGBASE,MEMBASE,IFLAG);
    aclear(IXS,IYS,NCOL,NROW,150);
    initialize();
    system("cls");           /* Clears monitor screen  */
}
/*.....*/

readit()
{
    int errval;
```

```
    int ch;

    while(1)
    {
        printf("\n\n\tENTER IMAGE FILENAME as (DEV:name.IMG)-->");
        fflush(stdin);
        gets(filename.s);
        filename.l = (char)strlen(filename.s);

        errval = readim(IXS,IYS,NCOL,NROW,filename,&comline);

        if( errval == 0 ){
            printf("\nFILE--> %s \n\nCOMMENT: %s ",filename.s,comline.s);
            break;
        }
        if( errval != 0 )
        {
            printf("\n\aError reading File!!");
            if( errval == -2 ) printf("\n\tFile Not Found!");
            if( errval == -3) printf("\n\t\tBad File Format!");
            printf("Try Again?? ...Yes (y) / No (n) ");
            ch = getch();
            if( ch ==  'Y' || ch ==  'y' ) continue;
            if( ch ==  'N' || ch ==  'n' ) exit(0);
        }
    }
}
/*.....*/

saveit()
{
    int comflag,errval;
    int ch;

    initialize();

    while(1)
    {
        printf("\n\n\tENTER FILENAME...( DEV: name.IMG )-->");
        fflush(stdin);
        gets(filename.s);
        filename.l = (char)strlen(filename.s);
        geti("\n Enter File Compression value ?...( 0/1 )  ",&comflag);
        printf(" \n\n Enter Image Comment or Return ..to continue.. (Max 200 CHAR):...");
        fflush(stdin);
        gets(comline.s);
        comline.l = (char) strlen(comline.s);

        if (comflag == 1) printf("\n\nStoring Image using Compression -- please wait!!");
        if (comflag == 0) printf("\nStoring Image -- plese wait!! ");

        errval = saveim( IXS,IYS,NCOL,NROW,comflag,filename,comline );

        if( errval == 0)
        {
            printf("\nImage Save completed SAT..");
            break;
        }
        if( errval != 0 )
        {
            printf("\n\a Error saving File!!");
            if( errval == -1 ) printf( "\nInsufficient Disk space" );
            printf("\n\tTry Again?? ...Yes (y) / No (n) ");
            ch = getch();
```

```
            if( ch == 'Y' || ch == 'y' ) continue;
            if( ch == 'N' || ch == 'n' ) break;
        }
    }
}

/*.....*/

speckle()
{
    extern int srow, scol;
    int  m, n, x, y, nn;
    long smax, smin;
    long sdata[NUM];

    unsigned long deviation, ssum;

    float  smean, slocal, stotal, tot;
    float  spklindex;

    if( srow >= NROW ) srow = NROW;
    if( scol >= NCOL ) scol = NCOL;

    tot = (long)srow * scol;

    /* Commence calculation   */

    printf("\n\n\tCalculating Speckle INDEX...");

    for ( n = 1 ; n < srow-2 ; n++)
    {
        for ( m = 1 ; m < scol-2 ; m++)
        {
            smax = 0;
            smin = 260;
            ssum = 0;
            nn = 0;

            for( y = ( n-1 ) ; y < (n+2) ; y++ )
            {
                for( x = ( m-1 ) ; x < ( m+2 ) ; x++)
                {
                    sdata[nn] = rpixel( x,y );
                    ssum += (long)sdata[nn];
                    if( smax < sdata[nn] ) smax = sdata[nn];
                    if( smin > sdata[nn] ) smin = sdata[nn];
                    nn++;
                }
            }
            deviation = smax - smin;
            if( ssum == 0 ) smean = 1;
            smean = ssum / NUM  ;
            slocal = (float)deviation / smean;
            stotal += slocal;
        }
    }
    spklindex = stotal / tot;
    printf("\n\n\tThe Calculated %d by %d speckle index is %f ",srow,scol,spklindex);
}

/*.....*/
/* Virtual Array Access Routines */

int init_v_array(filename,rec_size,filchar)
```

```
char *filename, filchar;
int rec_size;
{
    long size;
    FILE *f;
    f = fopen(filename,"wb");
    if (f != NULL)
    {
        size = 0;
        fwrite(&size,4,1,f);        /* write array size of zero */
        fwrite(&rec_size,2,1,f);    /* and array element size   */
        fwrite(&filchar,1,1,f);     /* and fill char            */
        fclose(f);                  /* to file header */
        return(1);
    }
    else
        return(NULL);
}

VACB *open_v_array(filename,buffer_size)
char *filename;
int buffer_size;
{
    VACB *v_array;
    char *buf_ptr;
    int i;
    char filchar;

    /* allocate virtual array control block */

    v_array = (VACB *) malloc(sizeof(VACB));
    if (v_array == NULL) return(NULL);

    /* open virtual array file */

    v_array->file = fopen(filename,"r+b");
    if (v_array->file == NULL)
    {
        free(v_array);
        return(NULL);
    };

    /* get array size and element size for control block */

    fread(&v_array->size,4,1,v_array->file);
    fread(&v_array->elsize,2,1,v_array->file);
    fread(&filchar,1,1,v_array->file);
    v_array->buf_elsize = v_array->elsize + 4;

    /* allocate buffer */

    v_array->buffer = (char *) malloc(v_array->buf_elsize * (buffer_size + 1));
    if (v_array->buffer == NULL) {
        fclose(v_array->file);
        free(v_array);
        return(NULL);
    };
    v_array->buf_size = buffer_size;

    /* set up blank_rec using the fill character in array header */
    /* for initializing new array elements */

    buf_ptr = v_array->buffer + v_array->buf_elsize * v_array->buf_size;
    v_array->blank_rec = buf_ptr + 4;
```

```
    for (i = 0; i < v_array->buf_elsize; i++)
        *buf_ptr++ = filchar;

    /* set record index negative for all buffer elements */

    buf_ptr = v_array->buffer;
    for (i = 0; i < v_array->buf_size; i++)
    {
        *((long *)buf_ptr) = -1L;
        buf_ptr += v_array->buf_elsize;
    };
    return(v_array);
}

void close_v_array(v_array)
VACB *v_array;
{
    int i;
    char *buf_ptr;
    long rec_index, file_offset;

    buf_ptr = v_array->buffer;

    /* flush buffer */

    for (i = 0; i < v_array->buf_size; i++)
    {
        /* check each element index */
        /* if element present; write it to disk */

        rec_index = *((long *)buf_ptr);
        if (rec_index >= 0)
        {
            file_offset = header + rec_index * v_array->elsize;
            fseek(v_array->file,file_offset,0);
            fwrite(buf_ptr + 4, v_array->elsize,1, v_array->file);
        };
        buf_ptr += v_array->buf_elsize;
    };
    free(v_array->buffer);              /* de-allocate buffer */
    fclose(v_array->file);              /* close array file   */
    free(v_array);                      /* de-allocate VACB   */
}

void *access_v_rec(v_array,index)
VACB *v_array;
long index;
{
    char *buf_ptr;
    int buf_index;
    long rec_index, temp_index;


    /* calculate buffer address of referenced element */

    buf_index = index % v_array->buf_size;
    buf_ptr = v_array->buffer + buf_index * v_array->buf_elsize;
    rec_index = *(long *)buf_ptr;

    /* if element present, return its buffer address */
    if (rec_index == index) return(buf_ptr + 4);

    /* if element doesn't exist, extend the file */
```

```
    if (index >= v_array->size)
    {
        fseek(v_array->file,0,2);
        for (temp_index = v_array->size; temp_index++ <= index; )
v           fwrite(v_array->blank_rec, v_array->elsize, 1, v_array->file);
        v_array->size = index + 1;
        fseek(v_array->file,0,0);
        fwrite(&v_array->size, 4, 1, v_array->file);
    };

    /* if buffer slot is occupied by another element, */
    /* save it to disk  */

    if (rec_index >= 0)
    {
        fseek(v_array->file, rec_index * v_array->elsize + header, 0);
        fwrite(buf_ptr + 4, v_array->elsize, 1, v_array->file);
    };

    /* read referenced element into buffer slot */

    fseek(v_array->file, index * v_array->elsize + header, 0);
    fread(buf_ptr + 4, v_array->elsize, 1, v_array->file);
    *((long *)buf_ptr) = index;

    /* return address of element in buffer */

    return(buf_ptr + 4);
}
/*.....*/
```

APPENDIX K

FORTRAN AND C COMPILER OPERATIONS

The following DOS batch files list the FORTRAN and C compiler directives used to compile all the FORTRAN and C programs using the optimizing compilers. All of these batch files use the recommended command line compile directives found in the ITEX/PC programmers manual [Ref. 6].

The FORTRAN batch files consist of *FCOMP.BAT, FLINK.BAT, FPACK.BAT*, and *FORALL.BAT.* The programs should be located in a directory named FORTRAN along with the program to be compiled. The batch files are executed using a DOS command that includes the name of the file to be compiled or linked. An example of its use is as follows:

FCOMP newid

which will compile the file *newid.for* located in the FORTRAN directory on the DOS path. The other batch files operate in a similar manner. The file FORALL.BAT combines the compile, link, and pack files into one evolution. This file is useful for compiling and linking programs known to be free of errors with just one DOS command.

File *FCOMP.BAT*

```
PATH=\FORTRAN;C:\FORT4\SRC;C:\FORT4;C:\FORT4\BIN;C:\FORT4\LIB;C:\FORT4\INCLUDE;
SET INCLUDE=C:\FORT4\INCLUDE
SET LIB=C:\FORT4\LIB
SET TMP=C:\FORT4\TEMP

FL /C /FSD:\FORTRAN\%1 /FCD:\FORTRAN\%1 %1.FOR
```

File *FLINK.BAT*

```
PATH=\FORTRAN;C:\FORT4\SRC;C:\FORT4;C:\FORT4\BIN;C:\FORT4\LIB;C:\FORT4\INCLUDE;
SET INCLUDE=C:\FORT4\INCLUDE
SET LIB=C:\FORT4\LIB
SET TMP=C:\FORT4\TEMP

LINK %1 %2 %3,,,ITEXPC.LIB FORTRAN.LIB;
```

File *FPACK.BAT*

```
PATH=\FORTRAN;C:\FORT4\SRC;C:\FORT4;C:\FORT4\BIN;C:\FORT4\LIB;C:\FORT4\INCLUDE;
SET INCLUDE=C:\FORT4\INCLUDE
SET LIB=C:\FORT4\LIB
SET TMP=C:\FORT4\TEMP
EXEPACK %1.EXE %1.TMP
```

```
COPY %1.TMP %1.EXE
ERASE %1.TMP
```

File *FORALL.BAT*

```
PATH=\FORTRAN;C:\FORT4\SRC;C:\FORT4;C:\FORT4\BIN;C:\FORT4\LIB;C:\FORT4\INCLUDE;SET
INCLUDE=C:\FORT4\INCLUDE
SET LIB=C:\FORT4\LIB
SET TMP=C:\FORT4\TEMP

FL /C /FSD:\FORTRAN\%1 /FCD:\FORTRAN\%1 %1.FOR

LINK %1 %2 %3,,,ITEXPC.LIB FORTRAN.LIB;
EXEPACK %1.EXE %1.TMP
COPY %1.TMP %1.EXE
ERASE %1.TMP
ERASE *.COD
ERASE *.LST
ERASE *.OBJ
ERASE *.MAP
ERASE *.BAK
```

The C compiler file *ACOMP.BAT* performs essentially the same evolutions as the FORTRAN *files but only one file is necessary* to accomplish all of the evolutions. Use of this file requires the program to be compiled to be in the same directory as the *ACOMP.BAT* file and to be on the path specified. The program will be compiled and, if successful, the link evolution will also occur. The defaults used with the compile command line achieve the generation of packed code automatically. All C programs in this thesis were compiled using this batch file.

File *ACOMP.BAT*

```
PATH=\MS_C\WRK;\MS_C\TMP;\MS_C\SRC;\MS_C\BIN;\MS_C\LIB;\MS_C\INCLUDE;C:\DOS
SET INCLUDE=\MS_C\INCLUDE
SET LIB=\MS_C\LIB
SET TMP=\MS_C\TMP

CL  /C /FS /ZE /G0 /ALND %1.C

IF NOT ERRORLEVEL 1  LINK /NOD /E %1, ,ITEXPC.LIB SUPPORT.LIB MLIBCE.LIB;
```

LIST OF REFERENCES

1. Redman, D. N., ***Image Analysis of Solid Propellant Combustion Holograms Using an Imageaction Software Package***, Master's Thesis, Naval Postgraduate School, Monterey California, June 1986.

2. Edwards, T. D., ***Implementation of Three Speckle Reduction Filters for Solid Propellant Combustion Holograms***, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1986.

3. Orguc, E. S., ***Automatic Data Retrieval from Rocket Motor Holograms***, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1987.

4. Imaging Technology Incorporated, ***PCVISION Frame Grabber Manual***, Imaging Technology Incorporated, 1985.

5. Imaging Technology Incorporated, ***The ImageAction User's Guide***, Imaging Technology Incorporated, 1985.

6. Imaging Technology Incorporated, ***The ITEX/PC Programmer's Manual***, Imaging Technology Incorporated, 1985.

7. Mircosoft Corporation, ***Microsoft FORTRAN Optimizing Compiler for the MS-DOS Operating System User's Guide***, Microsoft Corporation, 1987.

8. Microsoft Corporation, ***Microsoft FORTRAN 4.1 Optimizing Compiler Language Reference***, Microsoft Corporation, 1987.

9. Microsoft Corporation, ***Microsoft FORTRAN 4.1 Optimizing Compiler Microsoft Codeview and Utilities***, Microsoft Corporation, 1987.

10. Mircosoft Corporation, ***Microsoft C 5.0 Optimizing Compiler for the MS-DOS Operating System User's Guide***, Microsoft Corporation, 1987.

11. Microsoft Corporation, ***Microsoft C 5.0 Optimizing Compiler Language Reference***, Microsoft Corporation, 1987.

12. Microsoft Corporation, ***Microsoft C 5.0 Optimizing Compiler Microsoft Codeview and Utilities***, Microsoft Corporation, 1987.

13. Crimmins, T.R., ***Geometric Filter for Reducing Speckle***, Optical Engineering, v. 25, May 1986.

14. Statistical Graphics Corporation, ***STATGRAPHICS Statistical Graphics System User's Guide***, STSC, Inc., 1986.

15. Tichenor, Mark, "Virtual Arrays in C," ***Dr. Dobb's Journal of Software Tools***, #138, May 1988.

16. Lee, J.S., ***Speckle Suppression and Analysis for Synthetic Aperture Radar***, Optical Engineering, v. 25, May 1986.

17. Borland International, ***Turbo C User's Guide***, v.2.0, Borland International, 1988.

BIBLIOGRAPHY

Bolon, Craig, ***Mastering C***, SYBEX, Inc., 1986.

Hansen, Augie, ***Proficient C***, Microsoft Press, 1987.

Kernighan, Brian W., and Ritchie, Dennis M., ***The C Programming Language***, Second Edition, Prentice Hall, 1988.

Kernighan, Brian W., and Plauger, P. J., ***The Elements of Programming Style***, McGraw-Hill Book Company, 1978.

Schildt, Herbert, ***Advanced C***, Second Edition, Osborne McGraw-Hill, 1988.

Schildt, Herbert, ***C: Power User's Guide***, Osborne McGraw-Hill, 1988.

Shaw, Richard Hale, "Writing Optimal C: Part 1," ***PC Magazine***, v. 7, August 1988.

Shaw, Richard Hale, "Writing Optimal C: Part 2," ***PC Magazine***, v. 7, September 1988.

Waite, Mitchell, Prata, Stephen, and Martin, Donald, ***C Primer Plus***, Revised Edition, Howard W. Sams & Company, 1987.

INITIAL DISTRIBUTION LIST

	No. copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3. Department Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
4. Professor J. P. Powers, Code 62Po Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	4
5. Professor D. W. Netzer, Code 67Nt Department of Aeronautics Naval Postgraduate School Monterey, California 93943-5000	2
6. Air Force Astronautics Lab Attention: Captain M. Moser Edwards Air Force Base, California 93523	1
7. Space and Naval Warfare Systems Command Department of the Navy Attention: Code 32-2 Washington, D.C. 20363-5100	2
8. LCDR D. S. Kaeser 15239 Century Oak Road Salinas, California 93907-1120	2